ioProgrammo Anno XI - N°4 (113) RIPROGRAMMA IL TUO SMARTPHONE • AJAX SECONDO .NET • JAVASCRIPT MOSTRA I MUSCOLI • UN ASSISTENTE PER MIGRARE A SQL SERVER EDIZIONI MASTER

BEHAVIOUR DRIVEN DEVELOPMENT

TESTA LE TUE APPLICAZIONI IN MODO "SCIENTIFICO" E ASSICURATI CHE SIANO PRIVE DI BUG

VERSIONE PLUS RIVISTA+LIBRO+CD €9,90 — VERSIONE STANDARD RIVISTA+CD €6,90

ioProgrammo

PER ESPERTI E PRINCIPIANTI

Poste Italiane S.p.A Spedizione in A.P. • D.L. 353/2003 (conv.in L.27/02/2004 n.46) art.1 comma 2 DCB ROMA Periodicità mensile • APRILE 2007 • ANNO XI, N.4 (113)

Basato su windows mobile

RIPROGRAMMA IL TUO SMARTPHONE

Entriamo nei meandri dei dispositivi mobili e ti spieghiamo come modificare anche l'impossibile

Layout
"Sbrandizza" il tuo telefono e sostituisci il logo dell'operatore con quello della tua azienda

Applicazioni
Modifica i menu "standard" e il desktop con uno personalizzato per te

Provisioning XML
Un unico file per modificare le impostazioni di tutti i cellulari del tuo network aziendale

Al via il corso **JAVA STRUTS**
Imparalo e lavora subito nel mondo delle Web Business Application

WEB 2.0
È AMORE FRA AJAX E XSLT
Con il primo eviti il reload delle pagine, con l'altro controlli l'output. Coppia perfetta!

MOBILE
LA PHOTO GALLERY SUL CELLULARE
Sfruttiamo FLICKR, PHP e poco altro e ricerchiamo le immagini direttamente dal telefonino

C++
COME TI GESTISCO QUELLA STRINGA
Le tecniche che ti consentono di evitare "SQL Injection" e altri problemi legati al testo

.NET
UNA CHAT MOLTO SPECIALE
Realizziamola con Windows Communication Foundation: nato per comunicare!!!

msdn WEBCAST
4 NUOVI VIDEO
PER SCRIVERE CODICE CHE NON TEME ATTACCHI
- Code Access Security • ASP.NET Security
- Scrivere codice sicuro • ASP NET 2.0: Realizzare custom membership provider

AJAX SECONDO .NET

Arrivano le Microsoft Ajax Extensions: le librerie pensate da Microsoft per portare le applicazioni nel Web 2.0. Noi ti spieghiamo come usarle

WEB
JAVASCRIPT MOSTRA I MUSCOLI
Con Prototype bastano poche righe di codice e la tua Web Application è pronta

DATABASE
UN ASSISTENTE PER MIGRARE A SQL SERVER
Ecco 10 Semplici passi per portare i tuoi vecchi DB Access al nuovo engine di Microsoft

PATTERN
L'OGGETTO CHE NON FA NULLA
Evita errori di gestione della memoria usando il "Null Pattern Object"

SOLUZIONI Automi cellulari. Facciamo evolvere il software usando la stessa logica presente nel corpo umano...

EDIZIONI MASTER www.edmaster.it

Anno XI - N.ro 04 (113) - Aprile 2007 - Periodicità Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo
http://www.ioprogrammo.it

Direttore Editoriale: Massimo Sesti
Direttore Responsabile: Massimo Sesti
Responsabile Editoriale: Gianmarco Bruni
Vice Publisher: Paolo Soldan
Redazione: Fabio Farnesi
Collaboratori: R. Allegra, L. Buono, D. De Michelis, F. Grimaldi, E. Viale, F. Cozzolino, I. Venuti, O. Peli
Segreteria di Redazione: Rossana Scarcelli

Realizzazione grafica: Cromatika S.r.l.
Art Director: Paolo Cristiano
Responsabile grafico di progetto: Salvatore Vuono
Coordinamento tecnico: Giancarlo Sicilia
Illustrazioni: M. Veltri
Impaginazione elettronica: Francesco Cospite, Lisa Orrico, Nuccia Marra, Luigi Ferraro

Realizzazione Multimediale: SET S.r.l.
Realizzazione CD-Rom: Paolo Iacona

Pubblicità: Master Advertising s.r.l.
Via C. Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121207
e-mail advertising@edmaster.it
Sales Director: Max Scortegagna
Segreteria Ufficio Vendite: Daisy Zonato

Editore: Edizioni Master S.p.a.
Sede di Milano: Via Ariberto, 24 - 20123 Milano
Sede di Rende: C.da Lecco, zona industriale - 87036 Rende (CS)
Presidente e Amministratore Delegato: Massimo Sesti
Direttore Generale: Massimo Rizzo

ABBONAMENTO E ARRETRATI

ITALIA: Abbonamento Annuale: ioProgrammo (11 numeri) €59,90 sconto 21% sul prezzo di copertina di €75,90 · ioProgrammo con Libro (11 numeri) €75,90 sconto 30% sul prezzo di copertina di €108,90 Offerte valide fino al 30/06/07.

Costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 02 831212.

La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 02 83121206, oppure via posta a EDIZIONI MASTER via C. Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- cc/p n.16821878 o vaglia postale (inviando copia della ricevuta del versamento insieme alla richiesta);
- assegno bancario non trasferibile (da inviarsi in busta chiusa insieme alla richiesta);
- carta di credito, circuito Visa, Cartasì, o Eurocard/Mastercard (inviando la Vs. autorizzazione, il numero di carta di credito, la data di scadenza, l'intestatario della carta e il codice CVV2, cioè le ultime 3 cifre del codice numerico riportato sul retro della carta).
- bonifico bancario intestato a Edizioni Master S.p.A. c/o BCC MEDIOCRATI S.C.AR.L. c/c 0 000 000 12000 ABI 07062 CAB 80880 CIN P (inviando copia della distinta insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.

Sostituzioni: qualora nei prodotti fossero rinvenuti difetti o imperfezioni che ne limitassero la fruizione da parte dell'utente, è prevista la sostituzione gratuita, previo invio del materiale difettato.
La sostituzione sarà effettuata se il problema sarà riscontrato e segnalato entro e non oltre 10 giorni dalla data effettiva di acquisto in edicola e nei punti vendita autorizzati, facendo fede il timbro postale di restituzione del materiale.
Inviare il CD-Rom difettoso in busta chiusa a:
Edizioni Master - Servizio Clienti - Via C. Correnti, 1 - 20123 Milano

Servizio Abbonati:
☎ **tel. 02 831212**
@ **e-mail: servizioabbonati@edmaster.it**

Assistenza tecnica: ioprogrammo@edmaster.it
Stampa: Arti Grafiche Boccia S.p.a. Via Tiberio Felice, 7 Salerno
Stampa CD-Rom: Neotek S.r.l. - C.da Imperatore - Bisignano (CS)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma

Finito di stampare nel mese di Marzo 2007

Questo mese su ioProgrammo

NUOVI ORIZZONTI PER PHP

La notizia del mese è quella del rilascio di "Delphi per PHP". Codegear, l'azienda creata appositamente da Borland per supportare lo sviluppo di prodotti dedicati alla programmazione, ne ha appena annunciato la disponibilità. Delphi per PHP è un IDE RAD basato su VCL in tutto e per tutto identico a quelli a cui Borland ci ha da tempo abituato. Tradotto in poche parole: esiste una palette di componenti da trascinare su una form per costruire una pagina web dinamica. Lo stile è quello che già da tempo caratterizza ASP.NET per esempio. Perché questo annuncio è particolamente importante? Semplicemente perché PHP è il linguaggio più diffuso nella rete e sul mercato non esiste nessun prodotto che possa innalzare in maniera così efficace la produttività di un programmatore quanto può farlo un IDE RAD. Si apre dunque uno scenario innovativo sul fronte degli IDE. Codegear tenta di riguadagnare le posizioni che nel tempo aveva perso a favore di ambienti come ad esempio Visual Studio. E' anche vero che da sempre i programmatori PHP agiscono in ambienti *nix like e che sono abituati a lavorare con editor sofisticati che fanno della gestione del testo il loro punto di forza, ma che hanno ben poca struttura grafica. Bisognerà capire come il mercato reagirà all'introduzione di questo prodotto. Per ora annotiamo che il costo è di €219, perciò particolarmente accessibile alle piccole e medie imprese oltre che allo sviluppatore individuale, inoltre sono previsti particolari sconti per gli studenti. Il mercato di riferimento appare dunque chiaro. Il target è quello classico costituito da una pletora di sviluppatori PHP che da sempre affollano il web con prodotti di qualità nonostante vengano spesso sviluppati senza grosse organizzazioni alle spalle. Nonostante questo siamo sicuri che anche le organizzazioni dalle dimensioni generose avranno interesse verso questo prodotto

All'inizio di ogni articolo, troverete un simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre *\soft\codice* e *\soft\tools*) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it.

RIPROGRAMMA IL TUO SMARTPHONE

Entriamo nei meandri dei dispositivi mobili e ti spieghiamo come modificare anche l'impossibile

Layout
"Sbrandizza" il tuo telefono e sostituisci il logo dell'operatore con quello della tua azienda

Applicazioni
Modifica i menu "standard" e il desktop con uno personalizzato per te

Provisioning XML
Un unico file per modificare le impostazioni di tutti i cellulari del tuo network aziendale

pag. 16

004-005 editoriale 2-03-2007 12:46 Pagina 5

AJAX SECONDO .NET

Arrivano le Microsoft Ajax Extensions: le librerie pensate da Microsoft per portare le applicazioni nel Web 2.0. Noi ti spieghiamo come usarle

IOPROGRAMMO WEB

GRAFICA

SISTEMA

SISTEMA

RUBRICHE

DATABASE

SOLUZIONI

QUALCHE CONSIGLIO UTILE

I nostri articoli si sforzano di essere comprensibili a tutti coloro che ci seguono. Nel caso in cui abbiate difficoltà nel comprendere esattamente il senso di una spiegazione tecnica, è utile aprire il codice allegato all'articolo e seguire passo passo quanto viene spiegato tenendo d'occhio l'intero progetto. Spesso per questioni di spazio non possiamo inserire il codice nella sua interezza nel corpo dell'articolo. Ci limitiamo a inserire le parti necessarie alla stretta comprensione della tecnica.

http://forum.ioprogrammo.it

RIVISTA + LIBRO + CD-ROM in edicola

I contenuti del libro

Lavorare con JSP

JSP è uno dei linguaggi più utilizzati all'interno di architetture d'impresa. I motivi del suo successo sono molteplici. Molti sono legati alla sicurezza, altri alla sua flessibilità, in gran parte alla sua capacità di lavorare in ambienti distribuiti. Certamente non si tratta di un linguaggio di rapido apprendimento, tuttavia conoscerlo approfonditamente significa essere in grado di proporsi a tutta una serie di strutture che ne hanno fatto il fulcro della propria attività. Ivan Venuti in questo libro affronta aspetti avanzati legati alla programmazione JSP, partendo da note architetturali legate ai pattern per arrivare a JMX e a JSF. Più di ogni altra cosa il libro contiene una serie di riferimenti a framework, librerie, tecniche che ne fanno una risorsa utilissima per chi ogni giorno si trova ad affrontare il variegato mondo JSP. Al termine della lettura avrete uno sguardo d'insieme su ogni tecnica che in un qualche modo può migliorare il vostro lavoro

TECNICHE, TRUCCHI E CONSIGLI PER UTILIZZARE SUBITO E AL MEGLIO IL LINGUAGGIO DELLE AZIENDE

- **Pattern e metodi di programmazione**
- **Introduzione alle Java Server Pages**
- **L'architettura JMX**

RIVISTA + CD-ROM in edicola

EZPUBLISH 3.9.0

Il framework per la costruzione di CMS

Sebbene ad un primo sguardo, dopo l'installazione, potreste pensare di trovarvi davanti ad un normale sistema CMS, EXPublish è molto di più. Si tratta infatti di un completo framework RAD per la costruzione di applicazioni di gestione di contenuti molto specializzato. E' possibile definire nuove classi, nuovi formati per l'input, definire il workflow dell'applicazione, elaborare template molto accurati utilizzando smarty.

In buona sostanza, in EXPublish è previsto un editor visuale delle classi così come un completo modello ad eventi. Se avete bisogno di una pagina che sia compilata secondo determinati camp, non dovete fare altro che definire la classe che la rappresenta, sarà EZPublish a creare per voi la struttura e le altre componenti necessarie all'inserimento e alla persistenza dei dati. Nonostante questo potrete sempre modificare il codice secondo le vostre esigenze. Infine bisogna annotare che si tratta di un'architettura estendibile con plugin, per cui particolarmente flessibile. EZPublish è insomma di un prodotto molto evoluto con potenzialità straordinarie, particolarmente idoneo per le grandi aziende ma facilmente scalabile sia in alto che in basso. Un ultima nota va alla community di supporto, nutrita e disponibile.

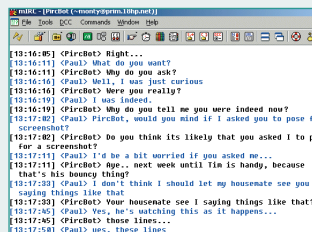
Directory:/Ezpublish

Prodotti del mese

Pircbot 1.4.4

Un bot irc programmabile in Java

Interessante questa idea di fornire agli amanti di IRC una serie di API che consentono di programmare un bot utilizzando Java. Normalmente chi si dedica a questo tipo di applicazioni utilizza Eggdrop e programma in TCL, in tutti e due i casi non si tratta di tecniche di larga diffusione, viceversa Java è un linguaggio che gode di un'immensa popolarità, per cui il poter programmare dei bot attraverso la flessibiltà di Java rappresenta senza dubbio un'opportunità interessante. Il metodo di programmazione è quello classico. Si progetta un bot che risponde a particolari sollecitazioni da parte degli utenti. Esistono però una serie di classi strutturate appositamente per gestire i trigger e gli eventi in risposta. Se volete divertirivi con un vostro BOT su IRC, dovete provarlo.

[pag.114]

Dadabik 4.2

Un creatore di interfacce verso database

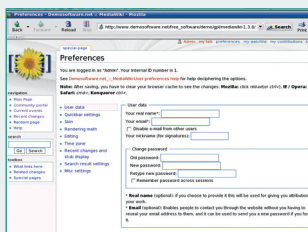
Si tratta di una web application scritta in PHP che consente di costruire altre web application basate su database. Ad esempio se volete costruire un sito che esponga un catalogo multimediale, non vi resta che informare Dadabik di quali campi si compone il catalogo in questione, di quali funzionalità volete che il vostro sito sia dotato, e lasciare a DadaBik il compito di generare la vostra interfaccia.Non è necessario avere competenze estese di programmazione, l'uso di DadaBik è piuttosto semplice. La flessibilità rimane comunque un punto di forza in Dadabik, di fatto a mezzo di form piuttosto autoesplicative si riescono a costruire interfacce molto sofisticate. Si tratta di un prodotto ottimo per progetti di medie dimensioni

[pag.115]

Mediawiki 1.9.2

La vostra wikipedia personalizzata

Un wiki è un sito internet gestito da qualcosa molto simile ad un CMS. Così come un CMS viene utilizzato per l'inserimento di rapido di contenuti sul Web, a differenza di un CMS però dispone di una sorta di linguaggio interno tale che se nel testo vengono inseriti dei marcatori particolari, automaticamente viene creata una pagina che li referenzia. Questo sistema garantisce la creazione rapida di enciclopedie che sfruttano al massimo il sistema di HyperLink tipico di internet. Inoltre le pagine vuote che vengono create automaticamente dal sistema possono essere riempite dagli utenti che ne hanno i permessi e così via con un meccanismo piramidale. Comprenderete che creare un'enciclopedia utilizzando questo sistema è piuttosto semplice. MediaWiki è il software che serve una delle più grandi enciclopedie mondiali online: wikipedia

[pag.115]

Irrlicht 1.2

Accendi il migliore motore 3D Open Source!

Irrlicht è un motore per la grafica tridimensionale, scritto in C++ e utilizzabile sia con questo linguaggio, sia con la tecnologia .NET.Presenta le principali caratteristiche di un motore professionale e vanta una notevole comunità di sviluppatori, con diversi progetti in attivo. Irrlicht ha tra i suoi pregi anche quello di potere utilizzare sia DirectX che OpenGL per cui diventa particolarmente adatto anche per lo sviluppo di giochi multipiattaforma. Nel tempo Irrlicht ha assunto dimensioni particolarmente rilevanti fino a diventare quasi uno standard per chi sviluppa VideoGames. I suoi connotati essenziali sono la curva di apprendimento molto rapida e la completezza delle funzionalità esposte, che consentono una flessibilità rara in questo mercato

[pag.115]

GLI ALLEGATI DI IOPROGRAMMO

Questo mese 4 Webcast indispensabili per imparare a scrivere applicazioni "blindate"

Code Access Security

Se dovessimo chiederci quale innovazione ha portato il .net Framework, la prima risposta sarebbe la Code Access Security, cioè la possibilità di controllare al run-time la pericolosità del codice che sta per essere eseguito. Dopo che i certificati digitali degli ActiveX hanno mostrato la loro inefficacia nel combattere il malware, viene dimostrata l'alternativa proposta dal Framework con gli smart client.

Speaker: Raffaele Rialdi

ASP .NET Security

ASP .NET offre una ricca infrastruttura per implementare l'autenticazione degli utenti in un'applicazione, siano essi appartenenti al dominio locale (Windows Authentication) oppure contenuti in una database (Forms Authentication). In questa session è spiegato inoltre come gestire i gruppi di Windows, i ruoli del Framework, il controllo "chirurgico" con la sicurezza imperativa e dichiarativa, i pro (ma soprattutto i contro) dell'impersonazione.

Speaker: Raffaele Rialdi

Scrivere Codice Sicuro

In questa sessione sono analizzati i motivi per cui è necessario scrivere codice "sicuro", quali sono gli attacchi più comuni e come ci si può difendere. Si parla principalmente di buffer overrun, di SQL injection e di Cross Site Scripting, valutando le pratiche di programmazione che è opportuno adottare per evitare problemi.

Speaker: Fabio Santini

ASP.NET 2.0: Realizzare custom membership provider

In ASP.NET 2.0 la forms authentication è costruita su un modello estensibile che permette di sostituire l'implementazione dei meccanismi di autenticazione senza richiedere la modifica del codice applicativo. Le "buzzword" inerenti questo modello sono "membership API" e "provider model", che sono presentate nel corso della session.

Speaker: Andrea Saltarello

PERCORSI FORMATIVI

msdn WEBCAST

Per chi desiderasse approfondire sono disponibili sul sito di Microsoft una serie di strumenti dedicati alla sicurezza

- **Securing Web Services**
- **Authentication & Digital Certificates**
- **"Non-Admin" Developing -**

Visita
http://www.microsoft.com/italy/msdn/risorsemsdn/security/default.mspx

FAQ

Cosa sono i Webcast MSDN?

MSDN propone agli sviluppatori una serie di eventi gratuiti online e interattivi che approfondiscono le principali tematiche relative allo sviluppo di applicazioni su tecnologia Microsoft. Questa serie di "corsi" sono noti con il nome di Webcast MSDN

Come è composto tipicamente un Webcast?

Normalmente vengono illustrate una serie di Slide commentate da un relatore. A supporto di queste presentazioni vengono inserite delle Demo in presa diretta che mostrano dal vivo come usare gli strumenti oggetto del Webcast

Come mai trovo riferimenti a chat o a strumenti che non ho disponibili nei Webcast allegati alla rivista?

La natura dei Webcast è quella di essere seguiti OnLine in tempo reale. Durante queste presentazioni in diretta vengono utilizzati strumenti molto simili a quelli della formazione a distanza. In questa ottica è possibile porre domande in presa diretta al relatore oppure partecipare a sondaggi etc. I Webcast riprodotti nel CD di ioProgrammo, pur non perdendo nessun contenuto informativo, per la natura asincrona del supporto non possono godere dell'interazione diretta con il relatore.

Come mai trovo i Webcast su ioProgrammo

Come sempre ioProgrammo cerca di fornire un servizio ai programmatori italiani. Abbiamo pensato che poter usufruire dei Webcast MSDN direttamente da CD rappresentasse un ottimo modo di formarsi comodamente a casa e nei tempi desiderati. Lo scopo tanto di ioProgrammo, quanto di Microsoft è infatti quello di supportare la comunità dei programmatori italiani con tutti gli strumenti possibili.

Su ioProgrammo troverò tutti Webcast di Microsoft?

Ne troverai sicuramente una buona parte, tuttavia per loro natura i webcast di Microsoft vengono diffusi anche OnLine e possono essere seguiti previa iscrizione. L'indirizzo per saperne di più è: http://www.microsoft.it/msdn/webcast/msdn segnalo nei tuoi bookmark. Non puoi mancare.

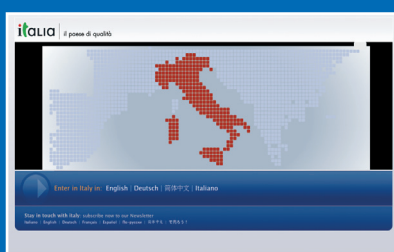
L'iniziativa sarà ripetuta sui prossimi numeri?

Sicuramente si.

News

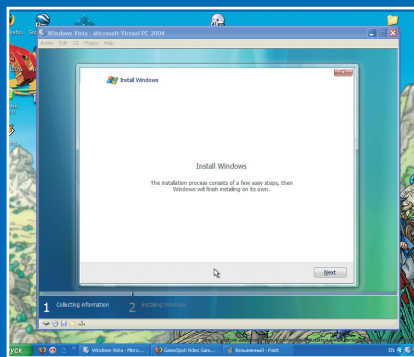
POLEMICA SU ITALIA.IT

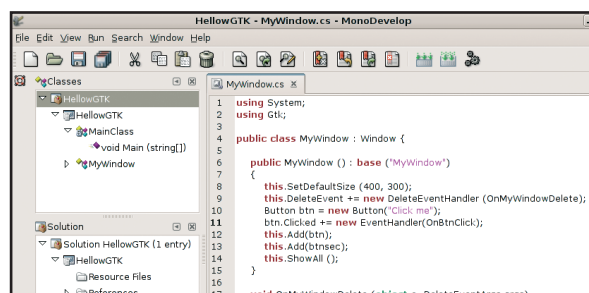
Da poco aperto al pubblico, il megaportalone governativo che dovrebbe rappresentare l'Italia nel mondo, ha già suscitato accese discussioni. Al centro del dibattito non solo il logo, sul quale si sono registrati parec-

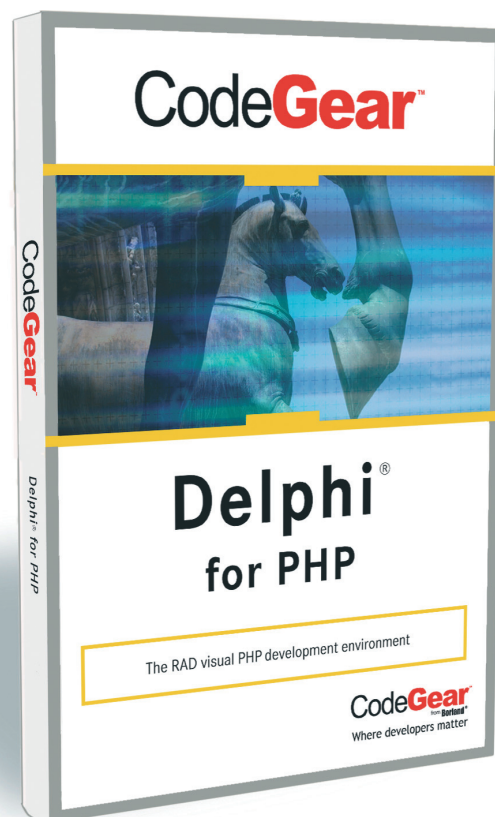
chi dissensi, ma anche e soprattutto l'accessibilità del portale. Nonostante esistano precise normative a riguardo della fruibilità dei contenuti anche per persone diversamente abili, il portalone governativo sembrerebbe disattendere molte delle norme previste.
A dire il vero la polemica è molto più complessa e coinvolge aspetti legali come tecnici. Resta il fatto che ci si sarebbe aspettata una maggiore attenzione da parte di un sito legato alle istituzioni rispetto a tali temi

WORD 2003 SUPPORTA ODF

Sun ha appena rilasciato un plugin per Microsoft Word 2003 che consente di leggere e scrivere file in formato OpenDocument.
Come molti sapranno ODF è un formato aperto sul quale si sta molto dibattendo negli ultimi tempi in relazione all'opportunità di rendere maggiormente trasparente il formato dei documenti. In realtà Microsoft aveva già rilasciato un analogo plugin con il supporto anche di Office XP e 2007. Il plugin rilasciato da Sun tuttavia è OpenSource. È interessante come il dibattito sull'OpenSource si stia spostando sui formati. Indipendentemente dal produttore, l'esistenza di tanto interesse è sintomatico di quanto il mondo dell'informatica stia rapidamente cambiando direzione

AL VIA VIRTUAL PC 2007

La virtualizzazione è l'ultima frontiera per quanto riguarda lo sviluppo di software tecnici. Su questo terreno negli ultimi anni si stanno consumando aspre battaglie. Da un lato c'è Microsoft con il suo Virtual PC, dall'altro un nutrito numero di concorrenti tutti dotati di caratteristiche tecniche di elevato livello, primo fra tutti VMware. È recente l'annuncio della disponibilità della nuova versione del virtualizzatore di Microsoft: Virtual PC 2007. Il prodotto è disponibile gratuitamente sul sito di MS e porta con se alcune interessanti novità. Prima di tutto il supporto a Vista, inteso sia come capacità di far girare sistemi Vista in una finestra virtuale sia come possibilità di far girare altri sistemi operativi in una finestra di Windows Vista. La seconda innovazione importante è relativa alla maggiore integrazione con i processori sia Intel sia AMD. Questo secondo Microsoft dovrebbe garantire prestazioni superlative. Una nota importante è relativa all'elenco dei sistemi supportati. Di fatto nella lista non compare alcun sistema Linux, questo probabilmente non significa che non si possano fare girare macchine virtuali di tipo *nix, ma che farlo significa non aspettarsi di avere il supporto della casa di Redmond. In ogni caso l'annuncio è interessante sia per le capacità tecniche del prodotto sia per le sue modalità di commercializzazione.

VISUAL BASIC CROSS PLATFORM

L'annuncio è sensazionale: il team di Mono, il noto progetto OpenSource per lo sviluppo di applicazioni .NET multipiattaforma, ha appena rilasciato un compilatore Visual Basic che consentirebbe di portare le applicazioni VB.net su Linux senza compiere nessuna modifica al codice sorgente. Come sempre, fra le intenzioni e la realtà dei fatti, bisogna saper trovare la giusta via di mezzo, tuttavia il passo è di quelli importanti. Esiste una schiera praticamente sterminata di sviluppatori Visual Basic ed una schiera altrettanto gigantesca di applicazioni VB sviluppate per il mondo Windows. Se il compilatore di Mono riuscisse a fare da ponte fra questi due sistemi si tratterebbe senza dubbio di uno dei più grossi successi della storia. I programmatori VB vedrebbero aprirsi un nuovo mercato che senza dubbio per dimensioni e potenziale di crescita sembra essere piuttosto appetitoso. Rimane da verificare quanto attualmente il compilatore sia affidabile e quanto il porting sia trasparente, tuttavia si tratta di un passo importante che da qui a qualche tempo potrebbe portare ad aprire scenari decisamente in-

ARRIVA DELPHI PER PHP

PHP è probabilmente il linguaggio più diffuso sul web. I suoi punti di forza sono noti: completezza, curva d'apprendimento rapida, flessibilità. È anche vero che tutti i concorrenti adottano ormai uno sviluppo RAD e visuale. A PHP fino a ieri mancava un IDE RAD. Oggi questa lacuna è stata colmata. Codegear, ovvero l'azienda creata appositamente da Borland per supportare i prodotti dedicati al mercato dello sviluppo ha appena immesso sul mercato “Delphi per PHP”. L'accoppiata è di quelle che creano entusiasmo negli sviluppatori. L'IDE di Delphi è da sempre uno dei più attrezzati rispetto alla programmazione RAD. Allo stesso tempo la VCL, ovvero l'insieme delle librerie che sottostanno all'IDE è decisamente stabile, completa e funzionale.
Cosa significa tutto ciò? Significa che i programmatori PHP possono da oggi iniziare a pensare per componenti. Creare una nuova pagina dinamica in molti casi si ridurrà a trascinare l'apposito componente su una form. L'ide provvederà a creare il codice necessario all'implementazione del controllo. Il codice sarà creato sulla base della VCL e pertanto godrà dell'ottima strutturazione a cui ci hanno abituata le librerie di Borland. L'approccio è decisamente diverso rispetto a quello al quale si era abituato chi da tempo produce codice PHP, tuttavia l'entusiasmo in relazione a questo progetto è già alto. Il prodotto è disponibile dal mese di Marzo ad un costo iniziale di 219†. È prevista anche una versione per le università e gli studenti con un costo decisamente inferiore. La VCL si basa su una serie di strumenti già noti quali Qooxdoo, Adodb, DynAPI, Smarty, XAjax e JSCalendar ed è completamente scritta in PHP. La struttura della libreria è dunque flessibile e garantisce che si possano scrivere nuovi componenti ed estendere quelli esistenti. Si tratta di un'innovazione importante che potrebbe radicalmente cambiare il modo di pensare al Web. La scommessa è di quelle che contano. Resta da vedere quanto i programmatori PHP siano disposti a rivedere il proprio modo di programmare

L'MP3 DELLA DISCORDIA

Sono 1,52 i miliardi che Microsoft dovrà versare ad Alcatel-Lucent per avere violato il brevetto sulla tecnologia di compressione alla base del formato MP3. La sentenza è da attribuirsi alla corte federale di San Diego ed è relativa ad un non autorizzato inserimento del supporto al formato MP3 all'interno del Windows Media Player sin dal 2003. In realtà la situazione è molto più controversa di quanto non sembri e coinvolge un numero di aziende impressionante e dai nomi piuttosto altisonante. A sviluppare il brevetto sul noto formato di compressione sono stati i laboratori congiunti del Fraunhofer Institute e i vecchi laboratori di Bell. Microsoft sostiene di avere acquistato proprio dal Fraunhofer Institute i diritti di sfruttamento della tecnologia per la modesta cifra di 16 milioni di dollari. Alcatel tuttavia che dal 2003 detiene la proprietà dei Bell Laboratories ha denunciato Microsoft per lo sfruttamento dei brevetti sulla console Xbox 360. Le due aziende avevano infatti stretto un accordo in relazione alla prima versione della console ma non alla seconda. La diatriba si è conclusa con una sentenza favorevole ad Alcatel. Tuttavia si tratta di un precedente preoccupante in quanto sono centinaia le aziende che hanno ottenuto il diritto di sfruttamento dal Fraunhofer Institute e fra esse compaiono nomi come Apple, Toshiba, Intel. Alcatel tuttavia sostiene la propria competenza sul brevetto relativo all'MP3 in quanto propositrice dell'idea originale che è poi stata concretizzata però dal Fraunhofer Institute. Si tratta dunque di una situazione controversa. Microsoft ricorrerà tuttavia in appello. Inutile aggiungere che pare proprio essere l'MP3 il formato più controverso dell'intero mondo IT ed in molti sensi.

MARCHIA IL TUO SMARTPHONE

IN QUESTO ARTICOLO IMPAREREMO COME PERSONALIZZARE I DISPOSITIVI MOBILI "BRANDIZZANDOLI" CON IL LOGO AZIENDALE. INOLTRE AGGIUNGEREMO APPLICAZIONI E MENU PERSONALIZZATI A WINDOWS MOBILE

Impegno

Tempo di realizzazione

Con i moderni dispositivi mobili è ormai consuetudine gestire le proprie mail o i propri appuntamenti "on the road". È consuetudine navigare su internet e rimanere connessi alla propria organizzazione anche quando si è fuori ufficio.
I dispositivi mobili possono essere utili anche per diverse applicazioni strategiche e possono essere utilizzati come "smart client" delle nostre applicazioni. In questo articolo impareremo a configurare gli Smartphone, basati su Windows Mobile, in modo da trasformarli in dispositivi "marchiati" con i loghi ed i brand della nostra azienda.
Impareremo come visualizzare il logo della nostra organizzazione all'accensione del dispositivo (come accade per i dispositivi marchiati dagli operatori mobili) e come mostrare una schermata *Home* (la scrivania dello smartphone) con tutti e soli i componenti utili agli utenti finali.

MICROSOFT DEVICE EMULATOR

Prima di iniziare la trattazione è importante spendere due parole su Microsoft Device Emulator che è una applicazione desktop che emula il comportamento dei dispositivi basati su Windows Mobile. Usando questa applicazione, potremo lanciare, testare ed eseguire il debug del nostro lavoro senza dover necessariamente possedere un dispositivo reale. Indubbiamente un emulatore si comporta "quasi" esattamente come un dispositivo reale ma può riservare qualche sorpresa quindi, per questa ragione, per un test definitivo dovremo distribuire e provare le nostre applicazioni, almeno su di un dispositivo.
In questo articolo tutti gli esempi verranno mostrati utilizzando Microsoft Device Emulator ed in particolare un dispositivo smartphone che monta Windows Mobile 5.0.

MICROSOFT DEVICE EMULATOR

Microsoft Device Emulator è compreso in Microsoft Visual Studio 2005 oppure è possibile scaricarlo dal sito Microsoft a partire dall'indirizzo http://msdn.microsoft.com/mobility/windowsmobile/downloads/default.aspx dove è possibile scaricare anche l'SDK di Windows Mobile 5.0 per Pocket PC e per Smartphone; questo componente aggiunge gli emulatori per la piattaforma Windows Mobile 5.0 al Device Emulator.

SCHERMATE DI AVVIO E DI ARRESTO

Ogni volta che accendiamo uno smartphone basato su Windows Mobile, possiamo vedere una sequenza di immagini o "splash screen". In genere appare prima il logo del costruttore (OEM - Original Equipment Manufacturer), quindi il logo del sistema operativo (Windows Mobile), infine può apparire un generico logo "Mobile Operator" o il logo di un operatore mobile specifico nel caso avessimo acquistato un dispositivo "marchiato". La stessa cosa può succedere allo spegnimento dello smartphone.
La prima personalizzazione che vedremo riguarda proprio la configurazione del dispositivo che ci consente di mostrare delle immagini personalizzate nelle fasi di accensione e spegnimento.
Avviamo Microsoft Device Emulator ed attiviamo uno smartphone basato Windows Mobile 5.0. Per avviare il dispositivo fare clic con il tasto destro del mouse sulla descrizione del dispositivo, quindi fare clic su *Connect*. Se non abbiamo mai avviato un dispositivo di questo tipo, possiamo vedere la sequenza degli "splash screen" (eccetto il logo del costruttore, dato che siamo in un ambiente emulato), così come riportato in **Figura 1**.
Se abbiamo già avviato questo tipo di dispositivo, non avremo la possibilità di vedere questa sequenza dato che l'emulatore salta la procedura di avvio su dispositivi già avviati. Per ovviare a questo comportamento dovremo operare un "soft reset" sul dispositivo. Così come mostrato in **Figura 2**. Facciamo clic sul menu *File* quindi

Fig. 1: La sequenza delle schermate all'accensione di un dispositivo smartphone.

facciamo clic su Reset, infine facciamo clic su *Soft*; l'applicazione chiede conferma in merito all'intenzione di effettuare il reset quindi dobbiamo confermare nuovamente facendo clic su si. Il dispositivo si oscura per qualche attimo quindi si riavvia mostrando, come atteso, la sequenza delle schermate.

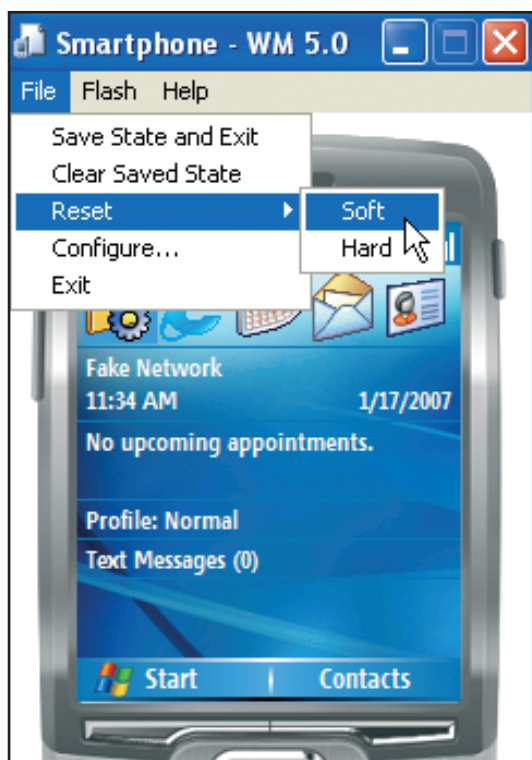



Fig. 2: Per visualizzare la sequenza di accensione su dispositivi già avviati è necessario effettuare un "soft reset"

PRODUZIONE DELLE IMMAGINI PERSONALIZZATE

Il nostro obiettivo è, ora, quello di rimpiazzare le immagini mostrate in Figura 1, con altre di nostra scelta che, presumibilmente, rappresenteranno il logo della nostra organizzazione.
Come prima attività, dobbiamo creare le immagini con le opportune dimensioni. Lo schermo del dispositivo smartphone emulato è pari a 176x200 pixel quindi dobbiamo creare delle immagini come quella mostrata, ad esempio, in **Figura 3**.

COPIA DELLE IMMAGINI NEL DISPOSITIVO

La seconda attività che ci aspetta è relativa alla copia nel dispositivo, delle immagini appena create. Nel caso del nostro emulatore, possiamo utilizzare Microsoft ActiveSync 4.1 che è una applicazione che consente la connessione tra dispositivi mobili (sia emulati che reali) basati su Windows Mobile ed il nostro computer.

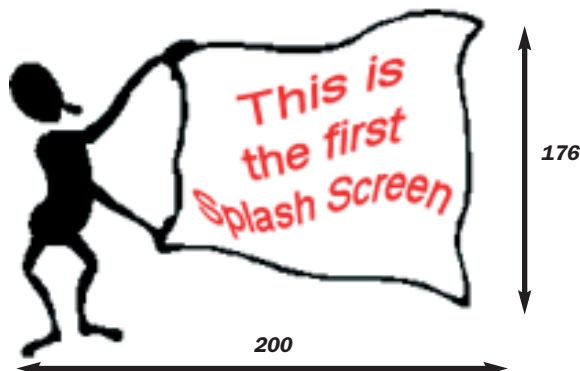



Fig. 3: Creiamo le nostre immagini personalizzate rispettando le dimensioni dello schermo.

Presupponendo di aver correttamente installato ActiveSync 4.1, attiviamo nuovamente il Device Emulator Manager, facciamo clic con il tasto destro sul dispositivo avviato, quindi facciamo clic su *Cradle*. Questa operazione stabilisce una connessione tra il PC ed il dispositivo. Ora, dall'applicazione ActiveSync possiamo esplorare il file system del dispositivo emulato, quindi possiamo copiare le nostre immagini, ad esempio, nella cartella *Application Data**Home* (si noti che sulla piattaforma Windows Mobile 2003 Second Edition per smartphone, la struttura delle cartelle è leggermente differente, in particolare il percorso diventa: *Storage**Application Data**Home*). La scelta della cartella non è vincolante ma nemmeno casuale nel senso che possiamo salvare le suddette immagini (nei file di supporto le due immagini che utilizzeremo si chiamano Splash1.gif e Splash2.gif) dovunque ma vedremo nel corso della trattazione che la cartella scelta ha un suo particolare utilizzo.

È possibile scaricare ActiveSync 4.1 da sito Microsoft a partire dall'indirizzo: http://www.microsoft.com/windowsmobile/downloads/activesync41.mspx).

Una volta salvate le immagini sul dispositivo, non ci resta che configurare lo stesso affinché mostri i nostri "loghi" in luogo delle immagini di default.

MODIFICA DEL FILE DI REGISTRO

Per questa ultima attività, abbiamo bisogno di modificare il registro del dispositivo. Abbiamo diverse possibilità. Ad esempio possiamo installare una applicazione di terze parti che ci consenta di lavorare direttamente sul dispositivo, altrimenti, se utilizziamo Microsoft Visual Studio 2005 come nostro ambiente di sviluppo, possiamo usare i *Remote Tools*, che sono un gruppo di applicazioni di utilità che ci consentono di interagire agilmente con i nostri dispositivi. In questo articolo la scelta ricade sul *Remote Registry Editor*, che appartiene , appunto, alla famiglia dei *Remote Tools*. Quando avviamo il *Remote Registry Editor*, la prima cosa che dobbiamo fare, è la scelta del dispositivo a cui connettersi (**Figura 4**); in questo caso scegliamo il Windows Mobile 5.0 Smartphone Emulator. Una volta connesso il dispositivo, dobbiamo aggiungere la chiave Splash Screen nel percorso *HKEY_LO-*

Fig. 4: Il Remote Registry Editor.

CAL_MACHINE\SOFTWARE\Microsoft (si noti che potremmo già trovare la chiave definita nel nostro dispositivo reale, nel caso questo fosse marchiato da un operatore mobile). Sotto la chiave *Splash Screen* dobbiamo aggiungere alcuni valori di tipo stringa seguendo le indicazioni della tabella 1. Come possiamo vedere dalla **tabella** precedente, se valorizziamo

Valore Stringa	Descrizione
MSBitmap	Percorso dell'immagine mostrata all'avvio in luogo del logo Windows Mobile
CarrierBitmap	Percorso dell'immagine mostrata all'avvio in luogo del logo Mobile Operator
CarrierTimeout	Se valorizzato a 0 non mostra la CarrierBitmap
MSShutdownBitmap	Percorso dell'immagine mostrata all'arresto in luogo del logo Windows Mobile
CarrierShutDownBitmap	Percorso dell'immagine mostrata all'arresto in luogo del logo Mobile Operator

Tabella 1

il parametro CarrierTimeout a zero ("0"), possiamo mostrare soltanto la prima schermata. In verità le schermate iniziali sono utilizzate per "coprire" l'avvio del sistema operativo; se proviamo ad "oscurare" la seconda schermata, in realtà non solo non accelereremo l'avvio in modo notevole ma inoltre vedremo lampeggiare alcune schermate di avvio che sicuramente non allieteranno i nostri utenti finali. Per ottenere un risultato ottimale è invece consigliabile mostrare sempre i due "tipi" di immagine; nel caso poi non avessimo due differenti tipi di messaggio da mostrare agli utenti, potremo comunque utilizzare la stessa immagine sia per sostituire il logo di Windows Mobile sia per sostituire il logo del Mobile Operator; il risultato sarà semplicemente la visualizzazione dell'immagine per tutta la fase del caricamento del sistema operativo. Nella **Figura 5** possiamo vedere, attraverso la schermata del *Remote Registry Editor*, come sono state valorizzate le stringhe per l'esempio che stiamo analizzando. Come detto abbiamo definito due immagini da mostrare all'avvio ed all'arresto del

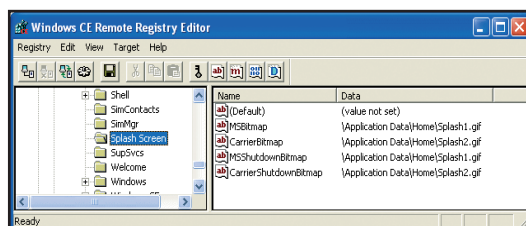



Fig. 5: Il registro del dispositivo è stato modificato per mostrare le nostre immagini all'avvio ed all'arresto del sistema.

sistema; se ora effettuiamo un "soft reset" così come descritto all'inizio della trattazione, possiamo vedere il risultato del nostro lavoro così come mostrato in **Figura 6**.

SCHERMATA HOME

La schermata *Home* dei dispositivi smartphone è paragonabile al desktop del nostro computer. Dalla

Fig. 6: Le nostre schermate vengono mostrate all'avvio del dispositivo

schermata *Home* possiamo avviare le applicazioni del nostro dispositivo senza dover necessariamente navigare la struttura a menù alla base dell'organizzazione delle applicazioni.
Gli smartphone sono dotati di una schermata *Home* di default che di solito è simile a quella mostrata in **Figura 7**; come possiamo notare, riusciamo ad avviare Pocket Internet Explorer, il Calendario piuttosto che l'applicazione di gestione del Messaging, semplicemente posizionandoci sulla porzione di schermo relativa all'applicazione (tecnicamente chiamata "plug-in"), e premendo il tasto *azione*.

***Fig. 7:* La schermata Home di defalut per il nostro dispositivo emulato.**

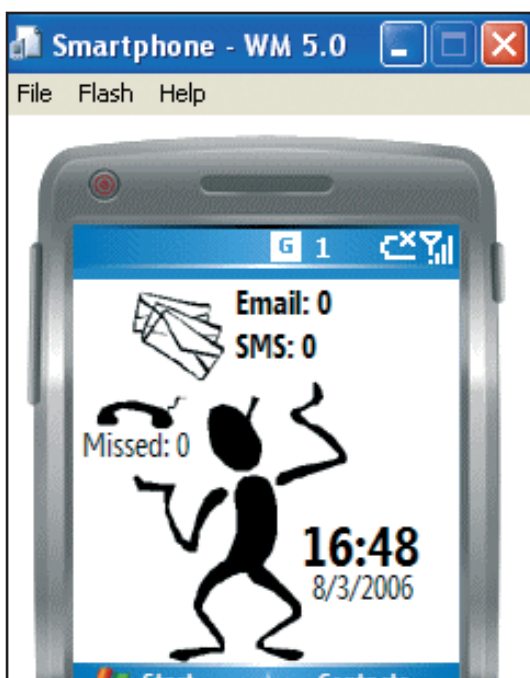



***Fig. 8:* Una schermata Home personalizzata.**

IL TASTO AZIONE

Ogni smartphone è dotato di un tasto azione che corrisponde al tasto sinistro del mouse per un computer da tavolo; nel caso del dispositivo emulato, il tasto azione è posto al centro dei tasti "4-vie" che consentono lo spostamento del cursore nelle quattro direzioni. Nei dispositivi reali, il tasto azione può assumere diverse forme; ad esempio in un modello commerciale abbastanza diffuso i tasti "4-vie" sono implementati mediante un cilindretto che consente un agevole navigazione alto/basso; in questo caso il tasto azione è rappresentato dallo stesso cilindretto e la pressione del tasto azione è ottenibile premendo il cilindretto stesso. In ogni caso il tasto azione è normalmente collocato in corrispondenza dei tasti "4-vie" quindi facilmente riconoscibile.

La schermata Home è composta da un layout, salvato in un file con estensione .home.xml e da una immagine di sfondo (un file in formato gif, jpg o bmp). Nel layout della schermata andremo a definire quali *plug-in* sono visualizzati, quali set di colori verranno utilizzati per visualizzare i componenti di sistema (menù, etichette, intestazioni delle maschere, ecc.), quale immagine di sfondo utilizzare ed, infine, qualche impostazione di default. L'obiettivo che ci poniamo è quello di realizzare una schermata *Home* come quella mostrata dalla **Figura 8**.

***Fig. 9:* L'immagine di sfondo per la nostra schermata Home.**

L'IMMAGINE DI SFONDO

Innanzitutto dobbiamo creare una immagine di sfondo, ad esempio come quella mostrata in **Figura 9**. Dobbiamo creare una immagine pensando a quali plug-in andremo a posizionarci sopra, di conseguenza dobbiamo immaginare dove e come andremo a visualizzare le informazioni rilevanti per i nostri utenti.
La creazione dell'immagine di sfondo in realtà è un processo ciclico dato che, a partire da una idea iniziale, solo dopo aver iniziato a posizionare i *plug-in* avremo una chiara idea sul risultato grafico finale. Per tale ragione è op-

portuno tenere sempre aperta l'applicazione di grafica con cui produrremo l'immagine di sfondo.

IL FILE DI LAYOUT

Come detto in precedenza, il layout viene specificato creando un file in formato XML.
Il documento ha come radice, l'elemento <*home*>, all'interno del quale è possibile identificare tre diversi frammenti che vengono utilizzati per definire rispettivamente: le informazioni generali della schermata *Home*, i valori di default e la lista dei *plug-in* da visualizzare.
Nella prima sezione possiamo aggiungere informazioni come il nostro nome, il nostro sito od il nostro indirizzo di posta elettronica. Ciò per consentire a chi fruirà della nostra schermata *Home*, di contattarci per eventuali osservazioni in merito.
L'elemento <*title*> è utilizzato per definire il nome della nostra schermata *Home*; è molto importante definire un valore significativo, dato che questo verrà visualizzato nella pagina di selezione delle schermate Home raggiungibile seguendo il percorso *Start->Settings->Home Screen*. Nella stesura del nostro file di layout, abbiamo l'opportunità di gestire la "localizzazione", ovvero l'automatico adattamento delle informazioni testuali visualizzate nella nostra schermata, alla lingua del dispositivo. Ad esempio, è possibile localizzare il titolo della nostra schermata, utilizzando l'attributo <*lang*> dell'elemento <*title*>. In particolare dovremo creare tante versioni del titolo quante sono le lingue che vogliamo supportare; ogni versione sarà caratterizzata dal valore esadecimale della lingua espresso tramite l'attributo <*lang*>. Nell'esempio seguente il titolo verrà mostrato nelle rispettive lingue se il dispositivo ha settata come lingua l'inglese, l'italiano, il francese o il tedesco.

```
<?xml version="1.0"?>
<home>
  <author>Your Name</author>

  <contacturl>Your website or your
                                   email</contacturl>
  <title lang="0x0409">My first Home</title>
  <title lang="0x0410">La mia prima Home</title>
  <title lang="0x040C">Mon premiere Home</title>
  <title lang="0x0407">Mein erster Home</title>
  <version>1.0</version>

…
<\home>
```

I VALORI DI DEFAULT

Per definire i valori predefiniti utilizzati nella nostra schermata *Home*, utilizziamo un secondo frammento del nostro documento di layout, che inizia con l'elemento <*default*>. Tramite l'elemento <*default*> possiamo definire, il font, la posizione relativa ed il formato del testo.
Nel frammento successivo, ad esempio, definiamo il font di default ed il formato del testo mostrato quando un plug-in è selezionato o non selezionato.

```
<default font-face="nina" font-size="14">
  <format state="unselected"
                              font-weight="normal" />
  <format state="selected" font-weight="bold" />
</default>
```

In questa sezione, tramite l'elemento <*background*>, definiamo anche quale immagine di sfondo utilizzare; se l'immagine è localizzata nella stessa cartella del documento di layout, possiamo specificare solo il nome del file; altrimenti dobbiamo necessariamente specificare il percorso completo per raggiungere l'immagine.

```
<background bgimage="home1.gif"
                align="bottom" />
```

SCHEMI DI COLORE

Gli schemi di colore (color schemes) sono usati per definire i colori ed i gradienti del dispositivo; ad esempio potremo definire il colore di sfondo delle applicazioni, oppure il colore del testo delle liste e delle etichette, oppure il gradiente della bara del titolo (definito tramite un colore di "inizio" ed uno di "fine").
Possiamo definire lo schema dei colori sia all'interno del file che definisce il layout, sia all'interno di un file XML separato (a cui daremo estensione .scheme.xml). Nel caso in cui si scelga di definire lo schema dei colori in un file separato, questo dovrà essere esplicitamente selezionato nella pagina di selezione della schermata Home, inoltre i valori contenuti in questo file sovrascriveranno i valori eventualmente specificati nel file di layout.
Tramite il frammento successivo definiamo il colore degli elementi selezionati in modo tale che l'elemento venga visualizzato con testo di colore bianco su sfondo di colore rosso (si veda la **Figura 10**).

```
<scheme>
 <color name="COLOR_HIGHLIGHT"
```

NOTA

GLI SCHEMI DEI COLORI

Una esaustiva descrizione di tutte le voci definibili in uno schema dei colori è consultabile nella libreria MSDN a partire dall'indirizzo:
http://msdn.microsoft.com/library/en-us/dnppcgen/html/sp02homescreen.asp

```
                                   value="#e80000"/>

  <color name="COLOR_HIGHLIGHTTEXT"
                                   value="#FFFFFF"/>

</scheme>
```

***Fig. 10:** Lo schema dei colori definito mostra gli elementi selezionati con testo bianco su sfondo rosso.*

SELEZIONIAMO I PLUG-IN

Nell'ultima parte del documento di definizione del layout dobbiamo listare i *plug-in* che vogliamo posizionare "sopra" l'immagine di sfondo.
I plug-in sono componenti che possono mostrare informazioni sulla schermata *Home*; alcuni di questi sono selezionabili e, di conseguenza, possono essere utilizzati per lanciare applicazioni; altri hanno solo una funzione informativa quindi non possono essere selezionati. All'interno del dispositivo esistono già alcuni *plug-in* che possono essere utilizzati immediatamente; questi sono relativi alle funzioni base del dispositivo; ad esempio esiste un *plug-in* per la parte di messaggistica, uno per il conteggio delle chiamate perse, uno per la visualizzazione della data e dell'ora, ecc. Nel caso avessimo esigenze particolari possiamo anche acquistare dei *plug-in* commerciali; sul mercato esistono *plug-in* per soddisfare praticamente ogni tipo di esigenza; da quelli che visualizzano le condizioni del tempo a quelli che visualizzano versi della Bibbia.

Nel caso, infine, in cui avessimo una esigenza talmente specifica da non trovare nulla di disponibile (o anche perché non vogliamo spendere soldi), come ultima possibilità, possiamo sviluppare il nostro *plug-in*. Allo stato attuale è possibile sviluppare *plug-in* solo in C++.
Nell'esempio attuale, posizioneremo alcuni *plug-in* disponibili nel dispositivo; questi vengono visualizzati nella schermata Home nello stesso ordine in cui compaiono nel file di definizione del layout e si estenderanno in un rettangolo di larghezza pari alla larghezza dello schermo e di altezza variabile.
Ogni plug-in è definito all'interno di un elemento <plugin> ed è identificato da un attributo CLSID.
L'altezza del plug-in è definita tramite l'attributo *height* e deve essere valorizzato pensando al tipo di informazione che verrà visualizzata.
Alcuni plug-in possono contenere elementi che definiscono variabili che vengono valorizzate dal sistema operativo dinamicamente. Ad esempio per il plug-in **Message Counter** l'elemento <unreadEmail/> verrà valorizzato con il numero di messaggi di posta elettronica non letti; l'elemento <unreadSMS/> verrà valorizzato invece con il numero di messaggi SMS non letti. Tramite l'elemento <label> definiamo il contenuto testuale del plug-in, che poi è l'informazione che verrà mostrata all'utente, definiamo inoltre la posizione relativa ed il formato del testo.
La posizione relativa del testo si riferisce all'angolo superiore sinistro del *plug-in* ed è modificata dai valori di default definiti nella prima parte del documento di definizione del layout (a meno di non sovrascrivere questi ultimi). Questo vuol dire che se abbiamo definito dei valori di "padding" nell'elemento *<default>*, il testo subirà un "padding" in modo consistente con i suddetti valori. Nel caso in cui non volessimo seguire i valori predefiniti, potremo sovrascrivere i valori definendone di altri contestualmente alla definizione del *plug-in*.

MESSAGE COUNTER

Nel seguente frammento, dove definiamo un plug-in di tipo Message Counter, mostriamo solo i messaggi di posta elettronica ed i messaggi SMS non letti. Aumentiamo la dimensione del font rispetto al valore di default che viene quindi sovrascritto; usiamo gli attributi x ed y dell'elemento <label> per disporre in maniera opportuna il testo.

```
<!-- Message Counter -->
<plugin clsid="{2F930BF0-6FE9-4a53-9E17-
                    88E9247BAB48}" height="60">
  <label y="5" x="75" halign="left"
       bgcolor="transparent" fgcolor="#000000" font-
                                          size="18">
    <text>Email: <unreadEmail /></text>
```

I PLUG-IN DEL SISTEMA

Una esaustiva descrizione di tutti i plug-in compresi nel sistema operativo e dei relativi elementi variabili è consultabile nella libreria MSDN a partire (nuovamente) dall'indirizzo:
http://msdn.microsoft.com/library/en-us/dnppcgen/html/sp02homescreen.asp

```
  </label>
  <label y="25" x="75" halign="left"
        bgcolor="transparent" fgcolor="#000000" font-
                                          size="18">
    <text>SMS: <unreadSMS /></text>
  </label>
</plugin>
```

Usiamo l'elemento <text> per giustapporre testo prefissato (come "EMail" o "SMS") e testo variabile (come <*unreadEmail* /> and <*unreadSMS* />), al fine di ottenere un informazione coerente da mostrare all'utente.
Il *plug-in* di tipo *Message Counter* è selezionabile quindi se premiamo il tasto azione quando il plug-in è selezionato, avviamo l'applicazione di gestione del Messaging.

MISSED CALLS

Nella definizione del successivo *plug-in* di tipo *Missed Calls*, vogliamo modificare il comportamento di default relativo alla selezione, ma, invece di sovrascrivere completamente i valori predefiniti, vogliamo aggiungerne di altri in modo che il *plug-in*, quando selezionato, mostri il testo in grassetto come da valori di default, ma anche in colore rosso.

```
<!-- Missed Calls -->
<plugin clsid="{0BA8ABB8-1F1D-417f-88C6-
                    DA8530E2E7A6}"  height="40">
  <format state="nocall" visible="true"/>
  <label y="12" x="2" halign="left"
       bgcolor="transparent" fgcolor="#000000" font-
                                          size="18">
    <format state="selected" fgcolor="#e80000" />
    <text>Missed: <calls/></text>
  </label>
</plugin>
```

In questo caso, all'interno dell'elemento <*label*>, dobbiamo definire un elemento <*format*> che estende il comportamento predefinito.
Anche il *plug-in* di tipo *Missed Calls* è selezionabile, ma potremo attivare l'applicazione di visualizzazione delle chiamate (*Call History*) solo se vi sono chiamate perse; nel caso in cui non ci siano chiamate perse, l'applicazione non viene lanciata.

DATE & TIME

Come ultimo *plug-in*, ne inseriamo uno di tipo *Date & Time*.
Definiamo le informazioni sull'orario per mezzo dell'elemento <*clock*> al cui interno porremo gli elementi per definire il tipo di informazione che vogliamo visualizzare. Nel caso in esame vogliamo mostrare sia l'orario che la data; con gli elementi <*time*> e <*date*>, possiamo definire la posizione, il formato e la modalità di visualizzazione rispettivamente di orario e data. L'orario verrà mostrato nella modalità "24 ore" mentre la data sarà espressa tramite cifre.

```
<!-- Date & Time -->
<plugin clsid="{E09043DF-510E-4841-B652-
    388316977A7A}" file="sysplug.dll" height="60">
  <clock>
   <time font-weight="bold" font-size="30"
 halign="left" fgcolor="#000000" mode="24" y="10"
                                        ="100" />
   <date font-weight="normal" fgcolor="#000000"
           font-size="17" valign="top" halign="left"
                mode="short" y="35" x="105" />
  </clock>
</plugin>
```

A questo punto abbiamo completato la compilazione del nostro file di layout che possiamo vedere nella sua forma integrale qui di seguito.

```
<?xml version="1.0"?>
<home>
  <!-- General information -->
  <author>Your Name</author>
  <contacturl>Your website or your
                               email</contacturl>
  <title lang="0x0409">My first Home</title>
  <title lang="0x0410">La mia prima Home</title>
  <title lang="0x040C">Mon premiere Home</title>
  <title lang="0x0407">Mein erster Home</title>
  <version>1.0</version>

  <!-- Default settings -->
  <default font-face="nina" font-size="14">
    <format state="unselected" font-
                               weight="normal" />
    <format state="selected" font-weight="bold" />
  </default>

  <!-- Background image -->
  <background bgimage="home1.gif"
                                valign="bottom" />

  <!-- Color scheme -->
  <scheme>
    <color name="COLOR_HIGHLIGHT"
                                value="#e80000"/>
    <color name="COLOR_HIGHLIGHTTEXT"
                                value="#FFFFFF"/>
  </scheme>

  <!-- PLUG-INS -->
  <!-- Iconbar -->
```

```
<plugin clsid="{837FC251-FE69-43ad-84E0-EBCEDEBA0884}" height="25">
  <iconbar fgcolor="#FFFFFF"/>
  <background gradient="title" />
</plugin>

<!-- Message Counter -->
<plugin clsid="{2F930BF0-6FE9-4a53-9E17-88E9247BAB48}" height="60">
  <label y="5" x="75" halign="left" bgcolor="transparent" fgcolor="#000000" font-size="18">
    <text>Email: <unreadEmail /></text>
  </label>
  <label y="25" x="75" halign="left" bgcolor="transparent" fgcolor="#000000" font-size="18">
    <text>SMS: <unreadSMS /></text>
  </label>
</plugin>

<!-- Missed Calls -->
<plugin clsid="{0BA8ABB8-1F1D-417f-88C6-DA8530E2E7A6}" height="40">
  <format state="nocall" visible="true"/>
  <label y="12" x="2" halign="left" bgcolor="transparent" fgcolor="#000000" font-size="18">
    <format state="selected" fgcolor="#e80000" />
    <text>Missed: <calls/></text>
  </label>
</plugin>

<!-- Date & Time -->
<plugin clsid="{E09043DF-510E-4841-B652-388316977A7A}" file="sysplug.dll" height="60">
  <clock>
    <time font-weight="bold" font-size="30" halign="left" fgcolor="#000000" mode="24" y="10" x="100" />
    <date font-weight="normal" fgcolor="#000000" font-size="17" valign="top" halign="left" mode="short" y="35" x="105" />
  </clock>
</plugin>
</home>
```

Per rendere attiva la nostra prima schermata *Home*, dobbiamo innanzitutto copiare il file appena prodotto sul nostro dispositivo. Utilizziamo ancora una volta Microsoft ActiveSync 4.1 e copiamo il file, che nel frattempo abbiamo chiamato *Home1.home.xml*, nella cartella *\Application Data\Home* (si noti che nel caso in cui si usi un dispositivo basato su Windows Mobile 2003 SE, la struttura delle cartelle è leggermente differente ed il percorso è:

Fig. 11: Sequenza di schermate per selezionare la nostra prima Home.

\Storage\Application Data\Home). Ora non ci resta che selezionare la schermata Home dall'applicazione Home Screen, seguendo il percorso *Start->Settings->Home Screen*, così come mostrato nella sequenza di **Figura 11**. Una volta selezionata la schermata, se torniamo alla pagina iniziale e se abbiamo seguito correttamente tutte le istruzioni, potremo effettivamente vedere la schermata che volevamo ottenere, ovvero quella mostrata precedentemente in **Figura 8**.

CONCLUSIONI

Dato che i moderni dispositivi mobili stanno diventando sempre più degli "smart client" delle nostre applicazioni, perché non marchiarli con i loghi della nostra organizzazione? La risposta a questa domanda è il punto di partenza di questo articolo dove abbiamo imparato come configurare un dispositivo smartphone per mostrare all'avvio ed all'arresto delle immagini personalizzate; abbiamo inoltre imparato a configurare la schermata *Home*, che rappresenta il desktop del dispositivo. In questo caso, una opportuna configurazione consente ai nostri utenti finali di utilizzare tutte e sole le applicazioni utili per il loro lavoro, evitando di dover navigare la complessa struttura a menù. Tecniche di questo genere non rappresentano certamente la logica di business di un'applicazione ma rendono più professionale ed elegante il lavoro svolto. Inoltre la personalizzazione del desktop a mezzo di plugin rappresenta un'ottima strada per integrare i dati provenienti dal nostro software in un'interfaccia il più possibile amichevole. Non dimentichiamoci che la programmazione dello strato di presentazione fa pur sempre parte del nostro lavoro.

Oscar Peli

PROVISIONING XML PER DISPOSITIVI MOBILI

IL VOSTRO CLIENTE VI HA CHIESTO DI FARE IN MODO CHE I TELEFONINI DELL'AZIENDA POSSANO ESSERE CONFIGURATI AUTOMATICAMENTE SCARICANDO UN SEMPLICE FILE DA UN SITO WEB. COME FARE? ECCO LA SOLUZIONE

REQUISITI

Conoscenze richieste
Conoscenze base di programmazione Xml

Software
Sistema operativo: Windows 2000/XP Professional e IIS. Visual Studio 2005

Impegno

Tempo di realizzazione

Con "Provisioning XML " si indica la possibilità di configurare dispositivi smartphone e pocket PC basati sul sistema operativo Windows mobile (sia versione 2003 SE che 5.0) a partire da documenti XML. In questo articolo daremo uno sguardo al provisioning XML dal punto di vista di un amministratore di dispositivi. Questa precisazione è importante dato che è possibile effettuare operazioni diverse se siamo "operatori mobili" o "costruttori" o semplicemente i gestori dei dispositivi mobili della nostra organizzazione.
Per effettuare il provisioning di un dispositivo Windows Mobile dovremo innanzitutto creare un documento XML che contiene le istruzioni di configurazione come, ad esempio, le connessioni GPRS piuttosto che i parametri per l'ActiveSync.
Quindi dovremo testare il file creato. nel caso in cui non disponessimo di un dispositivo di test useremo una applicazione desktop ovvero il Microsoft Device Emulator che simula (quasi) perfettamente il comportamento sia di dispositivi Smartphone che Pocket PC. Come ultima fase dovremo inviare il nostro file di provisioning ai dispositivi in attesa di configurazione; vedremo alcune possibilità tra cui poter scegliere per ottimizzare il processo globale di configurazione.
Esistono diverse possibilità per effettuare il provisioning di dispositivi basati su Windows Mobile; in questo articolo ci occuperemo del protocollo Open Mobile Alliance (OMA) Client Provisioning
In questo scenario il ruolo fondamentale è giocato dal Configuration Manager che è un componente del sistema operativo che elabora i file di provisioning che vengono inviati ai dispositivi.
Il Configuration Manager si occupa di invocare ulteriori componenti del sistema operativo, i Configuration Service Provider, che si occupano della gestione di particolari aspetti della configurazione.
Ogni file di provisioning contiene le istruzioni di configurazione espresse tramite il formato XML; il Configuration Manager invoca i relativi Configuration Service Provider, passa loro le istruzioni di configurazione e questi ultimi si occupano di variare o interrogare i valori di configurazione (nell'ambito del provisioning non ci serve sapere oltre riguardo a questi componenti).
Ci sono numerosi Configuration Service Provider attraverso cui è possibile configurare praticamente ogni aspetto del nostro dispositivo mobile: il *Registry* viene utilizzato per interagire con il registro di sistema; il *CM_GPRSEntries* serve per configurare le connessioni GPRS; tramite il *BrowserFavorite* gestiremo i preferiti di Pocket Internet Explorer. Nel corso dell'articolo vedremo alcuni Configuration Service Provider e con questi configureremo una connessione GPRS ed alcuni preferiti di Pocket Internet Explorer.

FILE DI PROVISIONING

Un file di provisioning è un documento XML che segue la DTD (document type definition) MSPROV che estende la PROV DTD.
In realtà tralasciando tutti i dettagli teorici, un file di provisioning è abbastanza semplice; la radice del documento è rappresentata dall'elemento *<wap-provisioning-doc>*; la radice contiene elementi *<characteristic>*, uno per ogni Configuration Service Provider che vogliamo invocare. Per meglio definire le istruzioni di configurazione è anche possibile utilizzare ulteriori sotto-elementi *<characteristic>*. Le foglie del nostro documento XML sono rappresentate da elementi *<parm>* che definiscono capillarmente il parametro da settare tramite la coppia di attributi *name* e *value*.
Di seguito possiamo vedere un esempio di documento di provisioning dove andremo a settare una connessione GPRS

```
<wap-provisioningdoc>
  <characteristic type="CM_GPRSEntries">
    <characteristic type="Connessione GPRS">
      <parm name="UserName" value="mioUser" />
      <parm name="Password" value="miaPass" />
      <parm name="DnsAddr" value="10.6.100.100" />
      <parm name="AltDnsAddr" value="10.6.100.101"
/>
```

```
    <parm name="WinsAddr" value="10.6.100.100"
                                           />
    <parm name="AltWinsAddr" value="10.6.100.101"
                                           />
    <parm name="SpecificNameServers" value="1" />
    <parm name="DestId" value=
  "{436EF144-B4FB-4863-A041-8F905A62C572}" />
    <characteristic type="DevSpecificCellular">
      <parm name="GPRSInfoAccessPointName"
                              value="mio.apn" />
    </characteristic>
   </characteristic>
  </characteristic>
</wap-provisioningdoc>
```

Nel primo elemento *<characteristic>* dobbiamo indicare quale Configuration Service Provider invocare e lo specifichiamo attraverso il parametro type; in questo caso il valore è *CM_GPRSEntries*.
All'interno del primo elemento *<characteristic>*, dobbiamo utilizzare un altro elemento dello stesso tipo per specificare il nome della connessione (prestiamo attenzione nella definizione del nome dato che il sistema operativo non consente la duplicazione dei nomi per le connessioni GPRS).
Di seguito dobbiamo definire alcuni parametri come lo *username* e la *password* per la connessione; in questo caso è importante notare che i dati di configurazione possono non essere codificati quando inviati al dispositivo via etere (Over-The-Air o OTA) come vedremo nel seguito; per questa ragione è importante essere consapevoli degli eventuali rischi per la sicurezza quando si inviano dati sensibili come le password.
Nell'esempio che stiamo considerando effettuiamo anche una configurazione "avanzata" ovvero specifichiamo gli indirizzi IP sia del DNS che del WINS (primario e secondario); il parametro *SpecificNameServers* viene utilizzato per indicare che si vogliono (appunto) specificare gli indirizzi suddetti.
Il valore dell'ultimo parametro *DestId* è usato come GUID e rappresenta l'identificativo univoco della rete a cui verremo connessi tramite la connessione che stiamo configurando. La tabella seguente mostra la corrispondenza tra GUID e nomi delle reti disponibili.

GUID	Rete
436EF144-B4FB-4863-A041-8F905A62C572	Internet
A1182988-0D73-439e-87AD-2A5B369F808B	Intranet
7022E968-5A97-4051-BC1C-C578E2FBA5D9	WAP
F28D1F74-72BE-4394-A4A7-4E296219390C	WAP Sicura

Nell'esempio precedente la rete a cui verremo connessi è Internet.
Per terminare la configurazione, dobbiamo, infine, usare un ulteriore elemento *<characteristic>* per definire i parametri GPRS utilizzati per inizializzare la connessione. Ad esempio, il parametro *GPRSInfoAccessPointName* ci consente di specificare l'Access Point Name (APN) che poi identifica il nome logico del gateway GPRS.
Per una completa descrizione di tutti i parametri disponibili, possiamo sempre far riferimento alla documentazione relativa allo specifico Configuration Service Provider che possiamo consultare a partire dall'indirizzo specificato in precedenza.

ELIMINAZIONE DEGLI ELEMENTI DI CONFIGURAZIONE

Quando acquistiamo un dispositivo marchiato, è possibile che il produttore abbia definito alcuni elementi di configurazione come alcuni preferiti di Pocket Internet Explorer piuttosto che alcune connessioni GPRS. In questa situazione è possibile che sia necessario rimuovere questi elementi per sostituirli con quelli corretti per la nostra organizzazione.
La DTD MSPROV definisce anche dei comandi che istruiscono il relativo Configuration Service Provider a ripulire il dispositivo da elementi di configurazione non necessari.
Per rimuovere particolari elementi di configurazione è possibile utilizzare nel documento di provisioning, elementi di tipo *<nocharacteristic>* e *<noparm>*.
Di seguito possiamo vedere un documento di provisioning dove innanzitutto vengono eliminati alcuni preferiti di Pocket Internet Expolorer, quindi vengono aggiunti quelli utili per la nostra organizzazione.

```
<wap-provisioningdoc>
  <characteristic type="BrowserFavorite">
   <nocharacteristic type="MSN Mobile"/>
   <nocharacteristic type="WindowsMedia.com"/>
   <characteristic type="Your Favorite">
     <parm name="URL"
                  value="http://www.yourfirst.com/"/>
     <parm name="Order" value="0"/>
   </characteristic>
   <characteristic type="Your Second Favorite">
     <parm name="URL"
                value="http://www.yoursecond.com/"/>
     <parm name="Order" value="0"/>
   </characteristic>
  </characteristic>
</wap-provisioningdoc>
```

Come possiamo vedere, è possibile eliminare ogni voce dei preferiti specificando il nome del relativo URL così come è mostrato nella lista dei preferiti di Pocket Internet Explorer.
Questo ovviamente significa che dobbiamo conoscere in anticipo tutti i nomi dei preferiti prima di poter-

li eliminare.Come ulteriore indicazione è importante notare che questo tipo di approccio funziona bene per dispositivi di tipo Smartphone ma non altrettanto per dispositivi di tipo Pocket PC. In particolare, i preferiti "built-in" dei dispositivi Pocket PC sono memorizzati in alcune chiavi del registro quindi anche se utilizziamo correttamente la procedura mostrata in precedenza, questo particolare tipo di preferiti non verrà eliminato.
Per eliminare anche questo tipo di preferiti dovremo utilizzare un approccio diverso ovvero un diverso Configuration Service Provider: il *Registry* Configuration Service Provider
Di seguito vediamo lo stesso documento visto in precedeza è stato modificato per lavorare correttamente con dispositivi Pocket PC; possiamo notare che in questo caso avremo la possibilità di eliminare tutti i preferiti con un'unica istruzione senza dover necessariamente conoscere in anticipo i preferiti da eliminare.

```
<wap-provisioningdoc>
  <characteristic type="Registry">
    <nocharacteristic
            type="HKCU\Software\Microsoft\Internet
                Explorer\Main\FavoritesEntries\"/>
  </characteristic>
  <characteristic type="BrowserFavorite">
   <characteristic type="Your Favorite">
     <parm name="URL"
            value="http://www.yourfirst.com/"/>
     <parm name="Order" value="0"/>
   </characteristic>
   <characteristic type="Your Second Favorite">
     <parm name="URL"
            value="http://www.yoursecond.com/"/>
     <parm name="Order" value="0"/>
   </characteristic>
  </characteristic>
</wap-provisioningdoc>
```

Ovviamente il documento precedente non funzionerà in dispositivi Smartphone; questo semplicemente perché in questa piattaforma non esistono le chiavi di registro con cui operiamo nella piattaforma Pocket PC.

PRODUZIONE DEL FILE CPF

Dopo aver completato la stesura del documento XML contenente le istruzioni di configurazione che vogliamo passare al nostro dispositivo mobile, dobbiamo innanzitutto salvare il file con il nome di *_setup.xml*.
Dopodichè dobbiamo rendere il file eseguibile in modo che il dispositivo sarà in grado lanciarlo ovvero di elaborarlo. Ciò che dobbiamo creare è un file CPF (CAB provisioning file) e per produrlo possiamo utilizzare il programma *makecab* che lanceremo da linea di comando dopo aver aperto un prompt dei comandi. I parametri da passare al programma sono i seguenti:

```
Makecab /D COMPRESS=OFF _setup.xml
                    smartConfigure.cpf
```

Si noti che dovremo usare l'opzione */D COMPRESS=OFF* solo per dispositivi Pocket PC.
Allo stesso modo potremo creare un file CAB "classico" utilizzando la stessa identica procedura ovvero utilizzando il seguente comando:

```
Makecab /D COMPRESS=OFF _setup.xml
                    smartConfigure.cab
```

La reale differenza tra il formato CPF ed il formato CAB sta nella modalità di esecuzione; innanzitutto in ambiente Windows Mobile 2003 SE il formato CAB, in questo caso, non viene riconosciuto come file valido quindi in questa situazione il formato CPF è l'unica soluzione. Altra fondamentale differenza è il colloquio utente-sistema che si instaura quando viene eseguito un file CPF piuttosto che un file CAB; ma questo concetto verrà chiarito più avanti nella trattazione dato che prima è necessario introdurre l'ambiente di test dei file di provisioning.

COME TESTARE I FILE DI PROVISIONING

E' pratica comune effettuare il test del proprio lavoro prima di effettuare la distribuzione su ambienti di produzione. Nel caso dei file di provisioning, lo scenario ideale sarebbe quello di possedere almeno un dispositivo per ogni tipo di sistema operativo che dovremo gestire e configurare. Nel caso non fossimo così fortunati, potremo utilizzare il Microsoft Device Emulator che è una applicazione desktop che emula il comportamento dei dispositivi basati su Windows Mobile. Se utilizziamo il Device Emulator possiamo eseguire, testare ed effettuare il debug dei nostri file di provisioning senza avere la necessità di possedere un dispositivo reale; certamente un emulatore si comporta "quasi" esattamente come un dispositivo reale quindi l'esito definitivo dell'esecuzione del nostro file di provisioning lo avremo solo dopo averlo eseguito almeno su di un dispositivo reale. Vedremo nel seguito un esempio di come l'emulatore si comporti in modo differente rispetto ad un dispositivo reale. Se abbiamo installato correttamente l'applicazione e tutti gli emulatori, una volta lanciato il Device Emulator Manager, vedremo qualcosa di simile a ciò che compare in **Figura 1**.

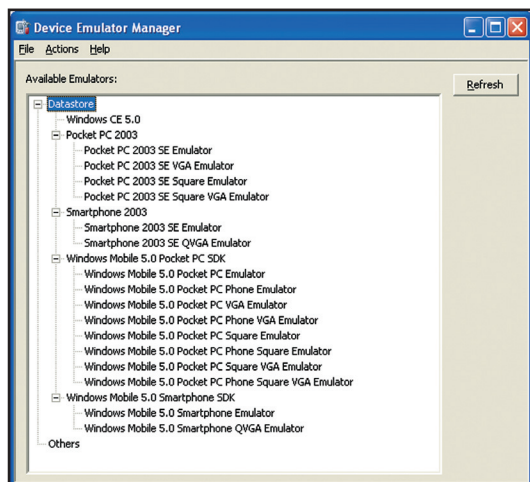



Fig. 1: Microsoft Device Emulator Manager

Per avviare un dispositivo, fare click con il tasto destro del mouse sulla descrizione del dispositivo quindi fare click su *Connect*; in **Figura 6** possiamo vedere un dispositivo Pocket PC 2003 Second Edition.
Per testare un file di provisioning dobbiamo innanzitutto inviarlo al dispositivo; vedremo le diverse possibilità di invio nella prossima sezione; in questo momento, per inviare il file all'emulatore, possiamo utilizzare Microsoft ActiveSync 4.1 che consente di connettere al nostro PC dispositivi mobili basati su Windows Mobile 5.0 e precedenti (sia reali che emulati).
Presupponendo di aver correttamente installato ActiveSync 4.1, attiviamo nuovamente il Device Emulator Manager, facciamo clic con il tasto destro sul dispositivo appena avviato, quindi facciamo click su *Cradle*. Questa operazione stabilisce una connessione tra il PC ed il dispositivo. Ora, dall'applicazione ActiveSync possiamo esplorare il file system del dispositivo emulato, quindi possiamo copiare il file di provisioning, ad esempio, nella cartella *My Documents*.
Supponiamo di utilizzare il file *smartConfigure.cpf* che avevamo prodotto in precedenza e che cancella i preferiti esistenti di Pocket Internet Explorer e ne

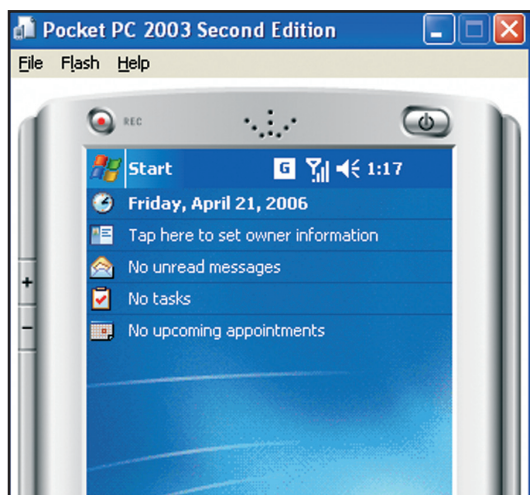



Fig. 2: Emulatore di un dispositivo Pocket PC 2003 Second Edition

aggiunge di nuovi; dopo averlo copiato nel dispositivo possiamo usare l'applicazione File Explorer (del dispositivo) per posizionarci nella cartella *My Documents* (**Figura 3**).
Per eseguire il file facciamo clic sul nome del file stesso; vedremo scomparire il file senza avere nessun messaggio esplicito di cosa sia successo. In verità se ci portiamo sulla schermata home vedremo che ci è stato recapitato un messaggio SMS. Se selezioniamo

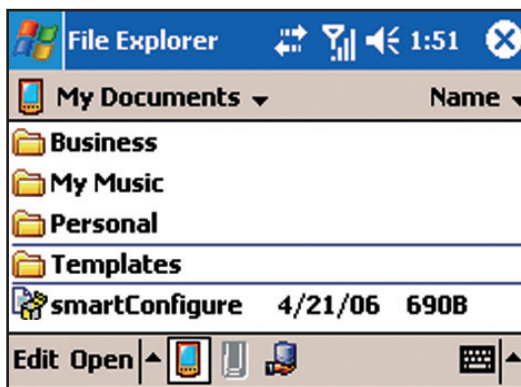



Fig. 3: Il file di provisioning smartConfigure.cpf è stato copiato nel dispositivo ed è pronto per essere eseguito

il messaggio ci rendiamo conto che il sistema ci informa che la configurazione è stata modificata con successo; la sequenza delle schermate è mostrata nella **Figura 4**.
Da notare invece che, in ambiente Windows Mobile 5.0, il file non scompare ma non si ha comunque nes-

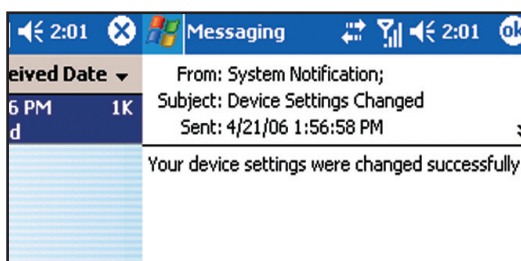



Fig. 4: Quando eseguiamo un file di provisioning in formato CPF, un messaggio SMS ci informa sull'esito dell'operazione

sun "segnale" dal sistema; il messaggio SMS viene invece recapitato regolarmente e ci informa sull'esito dell'operazione.
Dopo aver constatato che il file di provisioning è stato correttamente installato, possiamo verificare che in effetti siano state effettuate le modifiche che volevamo.
Lanciamo Pocket Internet Explorer e selezioniamo l'icona dei favoriti (la stella); la Figura 5 ci mostra che in effetti i preferiti attualmente presenti sono quelli che abbiamo definito nel nostro file di provisioning.
Da quanto detto finora abbiamo visto quanto sia facile testare i file di provisioning su piattaforma

Pocket PC; se usiamo gli emulatori per la piattaforma smartphone, le cose sono leggermente differenti. La differenza fondamentale per que-

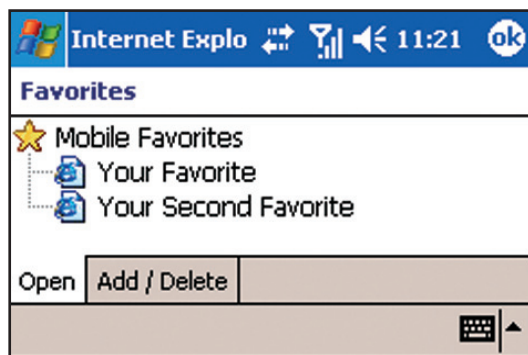



Fig. 5: Il file di provisioning ha modificato la lista dei preferiti

st'ultima piattaforma, è la mancanza di una applicazione tipo il File Explorer che ci ha consentito di eseguire i file CAB/CPF.

Per questa ragione dobbiamo inventarci un piccolo trucco per poter comunque verificare il nostro lavoro anche su piattaforma smartphone.

Innanzitutto dobbiamo attivare Internet Information Server (IIS) sul nostro PC; creiamo una directory virtuale e la chiamiamo, ad esempio, XML-Provisioning; poniamo in questa directory il nostro file smartConfigure.cab (usiamo il formato CAB per vedere cosa accade di differente rispetto ad un file CPF).

Attiviamo un dispositivo di tipo smartphone ed effettuiamo il "cradle" per connetterlo via ActiveSync; lanciamo Pocket Internet Explorer e selezioniamo la barra degli indirizzi; digitiamo la URL del file di provisioning quindi premiamo *Go*.

Se abbiamo digitato correttamente l'indirizzo un messaggio ci chiederà una conferma per scaricare il file; premendo *Yes*, il dispositivo scarica ed installa il file CAB (nella Figura 6 la sequenza delle schermate).

Possiamo notare che in questa occasione il sistema ci informa direttamente sull'esito dell'installazione.

Dobbiamo annotare questa ulteriore differenza

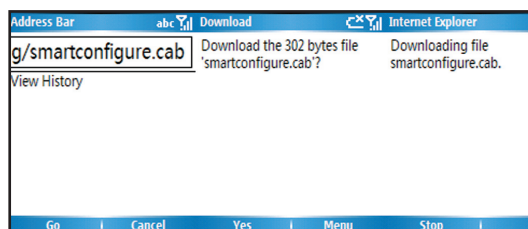



Fig. 6: Attraverso Pocket Internet Explorer è possibile scaricare ed installare file CAB.

tra file CAB e CPF in modo da poter decidere quale sarà la modalità migliore per la distribuzione dei nostri file di provisioning.

DIFFERENZE TRA EMULATORI E DISPOSITIVI REALI

La relativa facilità con cui abbiamo distribuito i nostri file di provisioning ai dispositivi emulati non ci deve trarre in inganno; questa situazione è permessa dall'ambiente "trusted" in cui ci stiamo muovendo; tutto accade all'interno del nostro PC dove (probabilmente) siamo anche amministratori di macchina, quindi, con il massimo dei diritti. In una situazione reale le cose non sono proprio così semplici dato che, nei dispositivi reali, esistono delle restrizioni di sicurezza che impongono un ulteriore sforzo per far funzionare le cose.

Ad esempio nei dispositivi basati su Windows Mobile 5.0, i file CPF non vengono eseguiti se non sono provvisti di una valida "signature" ovvero di un certificato che ne garantisca la sicurezza. Ciò implica sia un aumento del lavoro necessario per produrre un file che deve anche essere "segnato", ma soprattutto implica l'acquisto di un certificato da una delle autority di certificazione.

Se osserviamo la **Figura 7** possiamo notare che, a parità di file CPF, un dispositivo reale ha negato l'esecuzione (parte sinistra dell'immagine) mentre il corrispondente emulatore ha eseguito correttamente la configurazione (parte destra dell'immagine).

In questo caso, comunque, possiamo aggirare il

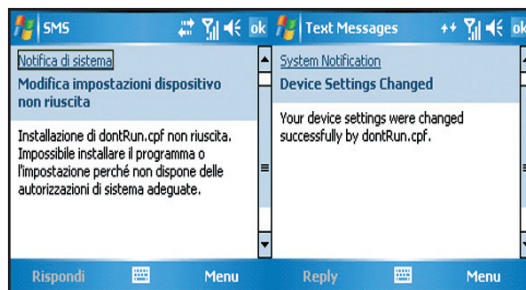



Fig. 7: Un dispositivo reale può avere un comportamento diverso rispetto al relativo emulatore

problema utilizzando file di tipo CAB che possono essere installati anche senza certificazione. Come possiamo vedere dalla **Figura 8**, se tentiamo di eseguire un file CAB, il sistema ci chiede conferma avvertendoci che il file è *fornito da un autore sconosciuto*; se accettiamo di continuare (premendo *Sì*) il sistema effettuerà l'installazione correttamente e la configurazione verrà aggiornata.

DISTRIBUZIONE DEI FILE DI PROVISIONING

Esistono diverse possibilità per inviare i file di provisioning ai dispositivi mobili.

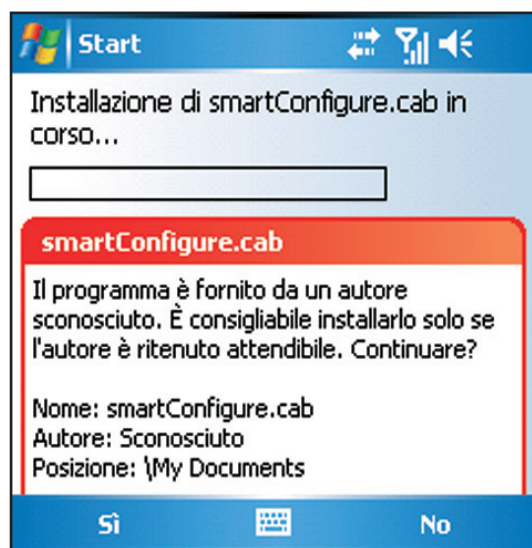



***Fig. 7:** Un dispositivo reale può avere un comportamento diverso rispetto al relativo emulatore*

Nel caso in cui il dispositivo sia "fisicamente" vicino alla postazione in cui produciamo i file, potremo utilizzare sia l'ActiveSync oppure la porta infrarossi (IR) o addirittura il Bluetooth se supportato. Normalmente questa situazione è possibile in fase di "bootstrap" ovvero la prima volta che scartiamo il dispositivo appena acquistato.
Certamente se la nostra organizzazione opera sul territorio è probabile che, una volta consegnati, i dispositivi restino nelle mani degli agenti; risulta quindi impossibile l'invio di ulteriori modifiche in modalità "connessa".
In questo caso l'unica alternativa è inviare i file via etere (Over-The-Air o OTA)
Abbiamo due possibilità di trasmissione via etere: "push" e "pull".
Nella modalità "WAP push", un documento che usa il formato definito nelle specifiche WAP Provisioning può essere "spinto" (push) ovvero inviato via etere, attraverso il meccanismo "WAP push" che utilizza il trasporto fornito dal Mobile Terminated Short Message Service (SMS). Purtroppo questa modalità è riservata agli operatori mobili che sono gli unici a possedere il PIN di rete del dispositivo (che non è il PIN che ci serve per sbloccare il dispositivo). Questa informazione è indispensabile per raggiungere il dispositivo e ciò impedisce l'utilizzo di questa modalità agli utenti finali. Nella nostra situazione, quindi, non ci resta che utilizzare la modalità "pull" dove è il dispositivo stesso che deve "tirare" (pull) a se il file. La soluzione consiste nel memorizzare i file di provisioning in un server web in modo tale che gli utenti possano utilizzare Pocket Internet Explorer per raggiungere il file come visto in precedenza.
Per facilitare l'operazione possiamo pensare di inviare ai nostri utenti un messaggio SMS contenente il link al file; in questo modo l'utente deve solo fare click sul link per avviare lo scaricamento del file e di conseguenza l'aggiornamento.
Tutti i meccanismi di distribuzione visti sinora soffrono di un problema comune ovvero la mancanza di persistenza. Indipendentemente dal modo con cui abbiamo trasmesso le informazioni di configurazione, se il dispositivo subisce un "hard reset" (ovvero viene riportato alla configurazione di fabbrica), delle nostre modifiche non resta traccia. Questo ovviamente perché il dispositivo ritorna alla configurazione che aveva non appena lo abbiamo acquistato quindi "pulito" da ogni successiva installazione.
Per ovviare a questo problema abbiamo due possibilità: la prima consiste nel porre i nostri file di configurazione direttamente nella ROM del dispositivo ovvero in quella parte del sistema che viene utilizzata per ristabilire la configurazione di fabbrica. In questo modo anche se il dispositivo viene sottoposto ad un "hard reset" i nostri file di provisioning vengono installati nuovamente in fase di ripristino del sistema. L'unica "pecca" di questa modalità sta nel fatto che è riservata solo ai produttori dei dispositivi stessi; tralasciando la questione di avere la possibilità di reperire gli strumenti e le procedure per aggiornare una ROM, il problema fondamentale sta nel fatto che, come utenti del dispositivo, non abbiamo semplicemente il diritto di modificare la ROM pena la perdita di ogni garanzia ed assistenza da parte del produttore del dispositivo.

UTILIZZARE LE SCHEDE DI ESPANSIONE

Abbiamo una seconda possibilità che ci consente comunque di ovviare al problema della persistenza. Se i dispositivi che dobbiamo gestire sono dotati di slot di espansione per schede Secure Digital (SD) o Multimedia Card (MMC), possiamo memorizzare i file di configurazione in una scheda e sfruttare la caratteristica di "Autorun" del dispositivo.
Questa caratteristica produce un risultato simile a ciò che accade normalmente nei nostri PC quando inseriamo un cd o un dvd ovvero viene eseguito automaticamente un programma. Per sfruttare questa caratteristica dobbiamo seguire i seguenti passi: innanzitutto dobbiamo sviluppare un semplice programma che esegue l'API *shellexecute* sul nostro file di configurazione; il programma dovrà essere nominato autorun.exe; quindi dobbiamo memorizzare il programma all'interno di una directory della scheda, nominata usando il valore numerico del processore che è pari a 2577 per processori StrongARM. Come ultimo

passo dobbiamo memorizzare il nostro file di provisioning in una cartella della scheda (anche la root) in modo che il programma autorun.exe possa raggiungerlo.
Ovviamente queste poche righe servono a dare solo una indicazione sulla modalità di utilizzo della caratteristica "Autorun" a cui sarebbe necessario dedicare un intero articolo per sviscerare tutte le complessità e gli aspetti del problema.
Tra le varie possibilità per distribuire i file di provisioning, probabilmente la soluzione migliore risulta essere quella di fornire a tutti i dispositivi una scheda SD o MMC per garantirsi la configurazione di base anche a fronte di "hard reset" del dispositivo.
Per fornire aggiornamenti successivi, la migliore modalità è quella di inviare un messaggio SMS ad ogni dispositivo in cui poniamo il link al file di provisioning; tramite Pocket Internet Explorer ogni utente potrà scaricare ed installare il file.

UN ULTIMO TRUCCHETTO

Tramite il programma Microsoft Device Emulator possiamo simulare anche il comportamento dei dispositivi a fronte di un provisioning basato sull'invio di messaggi SMS.
Lanciamo un emulatore, ad esempio un dispositivo Smartphone 2003 Second Edition; apriamo l'applicazione Messaging e scegliamo Text Messages. Scriviamo un nuovo messaggio SMS con il link al nostro file di provisioning quindi inviamolo al numero 14250010001.
In questo modo ci siamo auto-inviati un messaggio SMS ed abbiamo la possibilità di aprilo, selezionare il link e verificare che la procedura di aggiornamento funzioni correttamente. La **Figura 9** mostra la sequenza delle schermate ottenute in fase di testing.

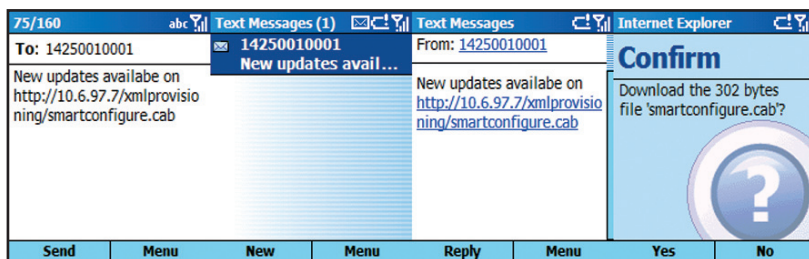



Fig. 9: Il numero 14250010001 ci consente di auto-inviarci messaggi SMS.

CASI DI STUDIO

Gli esempi d'uso del provisioning XML sono veramente moltissimi. Il più eclatante è probabilmente quello presente sul sito 190.it di Vodaphone dove è possibile configurare il proprio telefono semplicemente attraverso l'invio di un SMS. In ogni caso quasi tutti i provider di fornizi per dispositivi mobili consentono in una qualche maniera l'autoconfigurazione del telefonino. Per l'amministratore di una rete mobile il provisioning XML rappresenta l'unico modo di ottenere il completo controllo del proprio networking senza dover fisicamente disporre del dispositivo per poter effettuare le proprie personalizzazioni. Si tratta perciò di una risorsa particolarmente importante.

CONCLUSIONI

Tramite il provisioning XML possiamo configurare con semplicità dispositivi mobili basati su Windows Mobile 2003 Second Edition e 5.0.
Se adottiamo il protocollo Open Mobile Alliance (OMA) Client Provisioning, dobbiamo produrre un file XML dove indichiamo tutte le voci di configurazione che vogliamo aggiungere, modificare o cancellare. Quindi dobbiamo rendere eseguibile il file prodotto trasformando il documento XML in un file CAB o CPF; diverse sono le implicazioni che possono far preferire un formato piuttosto che un altro.
Infine dobbiamo distribuire i file di configurazione ai diversi dispositivi; anche in questo caso sono diverse le possibilità di invio; un mix di queste ci potrà garantire una certa tranquillità sul fatto di avere dispositivi sempre ben configurati anche se distribuiti sul territorio.
La parte più complessa del lavoro è rappresentata probabilmente proprio dal testing del file di provisioning. Considerando che è sempre buona norma provare il nostro lavoro prima di distribuirlo in un ambiente di produzione, con il programma Microsoft Device Emulator potremo effettuare il test dei file di configurazione.
Un'ultima nota deve essere dedicata alle differenze che spesso si incontrano nell'utilizzo di dispositivi reali. Nonostante Windows Mobile sia uno standard, spesso e volentieri i vari dispositivi non si comportano esattamente come ci attenderemmo, per cui al di là del test effettuato tramite l'emulatore presente in Visual Studio, è comunque importante effettuare un test almeno su due o tre dispositivi reali prima di entrare in produzione. Se si pensa che un file di provisioning può servire un elevato numero di dispositivi si comprende l'importanza di ridurre al minimo le eccezioni possibili, anche se dato l'infinito numero di periferiche mobili oggi disponibili, questa operazione contiene sempre un certo margine di rischio.

Oscar Peli

AJAX E XSLT COPPIA VINCENTE

DA UN LATO RISOLVIAMO IL PROBLEMA DELL'AGGIORNAMENTO DELLE PAGINE, DALL'ALTRO QUELLO DELLA VISUALIZZAZIONE DEI CONTENUTI. SI TRATTA DI DUE TECNOLOGIE CHE UNITE INSIEME APRONO SCENARI FINO AD ORA INIMMAGINABILI. VEDIAMO COME USARLE

Talvolta si associa a XML la fama di tecnologia utile soltanto in relazione alla sua particolare attitudine a immagazzinare e gestire, in modo semplice e ordinato, informazioni di vario tipo. La scelta di XML come database nasce infatti, principalmente, dalla necessità di trovare uno standard facilmente interfacciabile con qualsiasi dispositivo, oltreché semplice da interpretare anche da parte di chi non ha conoscenze particolari nel campo dell'informatica. In realtà, tuttavia, le potenzialità di XML sono ben superiori a quelle appena descritte, e nel prossimo futuro molto più che adesso ce ne renderemo sicuramente conto. **XML**, prima di tutto, si sposa perfettamente con quella che può considerarsi la grande "scoperta" del web moderno, **Ajax** (Javascript asincrono): un nuovo modo di concepire la programmazione *web-oriented*, basato su transazioni asincrone tra client e server. L'utilizzo di Ajax abbinato a XML consente di formattare con facilità la risposta HTTP generata dal server, e di effettuare in modo altrettanto semplice le più disparate interrogazioni da parte del client.
Ma XML è alla base, anche, di una serie di nuove tecniche di elaborazione dei dati, accomunate tutte quante da un unico principale obiettivo: favorire la fruizione di qualsiasi tipo di informazione da parte di una molteplicità di dispositivi, anche molto diversi tra di loro da un punto di vista tecnologico (palmari, portatili, browser vocali e per disabili, ecc...). Una di queste è XSLT, la tecnologia che prenderemo in considerazione in questo articolo: essa consiste nell'associare ad un file XML un foglio di stile particolare (un file con estensione XSL) che, a mezzo di un programma di *parsing*, consente la generazione di file in altri formati (sempre, comunque, legati a XML, come ad esempio **XHTML**).
Questo articolo, ovviamente, non ha certo l'ambizione di spiegare in modo completo e dettagliato tutti gli aspetti più "nascosti" di XML e dei suoi derivati, anche perché di sicuro non basterebbe un libro intero. Ciò che esso intende offrire, invece, è un semplice "assaggio" delle più interessanti funzionalità di XML (nel caso particolare, XSLT abbinato ad Ajax), al fine di fornire validi spunti per un approfondimento autonomo di tali argomenti.

XSLT: UN PO' DI TEORIA

XSLT sta per "Extensible Stylesheet Language Transformations", ed è stato introdotto come standard web dalla direttiva (*Recommendation*) W3C del 16 novembre 1999.
Il suo scopo principale è, come già accennato più sopra, quello di trasformare un file XML in un altro formato (XHTML, WML, SVG). Il file che definisce le regole in base alle quali deve avvenire la trasformazione è un foglio di stile XSL, caratterizzato da una rigida struttura XML e corredato anche da una serie di utili funzioni per l'elaborazione dei dati.
In altre parole, XSL è un vero e proprio linguaggio, dotato di blocchi condizionali, metodi per effettuare elaborazioni cicliche, ecc...
Per poter applicare delle regole o delle funzioni su determinati tag del file XML, occorre prima di tutto selezionare tali elementi utilizzando l'attributo *match*.
Il programma che effettua la trasformazione è un processore XSLT, il quale, durante l' operazione di *parsing* del file XML, non appena incontra una corrispondenza applica le regole contenute nel *template* (modello) associato a tale elemento, definito all'interno del file XSL.
Nell'esempio principale di questo articolo vedremo come realizzare un'applicazione che, al primo accesso, scarica un file XSL in locale, che verrà poi utilizzato ad ogni successiva richiesta remota per trasformare il codice XML ritornato dal server in XHTML.
I punti di forza di un'applicazione di questo tipo sono facilmente comprensibili: da un lato, lo

spostamento sul client della fase di elaborazione grafica e stilistica dei dati, mentre il server si limita a generare il contenuto base della risposta in formato XML, a tutto vantaggio della riduzione del carico di lavoro remoto. Dall'altro, la netta separazione delle logiche di elaborazione dei dati da quelle di presentazione e visualizzazione, che possono quindi essere studiate *ad hoc* per ogni singolo *media* di riferimento del sito.
Tra i difetti di questa soluzione, invece, spicca la necessità di disporre del processore XSLT direttamente sul client. Se il browser non è equipaggiato con il supporto a XSLT, è necessario, pertanto, implementare la stessa funzionalità lato server.

DALLA TEORIA ALLA PRATICA

Ma vediamo ora di affrontare qualche semplice caso pratico, per meglio inquadrare i reali vantaggi apportati da XSLT nell'attività quotidiana dello sviluppatore.
Immaginiamo di avere un file XML contenente le seguenti informazioni:

```
<?xml version="1.0" encoding="UTF-8"?>
<statistiche>
  <categorie>
    <capo>pantaloni</capo>
    <vendite>20</vendite>
  </categorie>
  <categorie>
    <capo>sciarpe</capo>
    <vendite>15</vendite>
  </categorie>
  <categorie>
    <capo>giubbotti</capo>
    <vendite>15</vendite>
  </categorie>
  <categorie>
    <capo>scarpe</capo>
    <vendite>50</vendite>
  </categorie>
</statistiche>
```

Se il nostro capo ci chiede di generare una tabella HTML sulla base del contenuto di tale file, possiamo utilizzare il seguente file XSL:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>
<xsl:template match="/">
  <html>
  <body>
    <table border="1">
      <tr bgcolor="yellow">
          <th>Capo</th>
          <th>Vendite</th>
      </tr>
    <xsl:for-each select="statistiche/categorie">
      <tr>
          <td><xsl:value-of
                  select="capo"/></td>
          <td><xsl:value-of
                  select="vendite"/>%</td>
      </tr>
    </xsl:for-each>
    </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Naturalmente dobbiamo creare un'associazione tra i due file, inserendo la seguente riga nel file XML:

```
<?xml-stylesheet href="statistica.xsl"
                    type="text/xsl"?>
```

Aprendo ora il file XML con un browser che supporta XSLT, vedremo come per magia il seguente risultato:

capo	vendite
pantaloni	20%
sciarpe	15%
giubbotti	15%
scarpe	50%

È bene, a questo punto, rimarcare che potremmo anche linkare un file CSS al file XML, e vedere il contenuto di quest'ultimo formattato in base alle nuove regole di stile, ma XSL è in grado di fornire una maggiore flessibilità offrendoci anche metodi di elaborazione dei dati (cicli, espressioni condizionali, ecc...).
Il nostro capo ci può chiedere anche, però, di elaborare un grafico con lo stesso file XML. Niente paura, basta creare un file XSL che genera un contenuto SVG, invece che XHTML, sempreché ovviamente il nostro browser supporti correttamente tale formato. Ecco il file in questione:

```
<?xml version='1.0'?>
<?xml-stylesheet href="" type="text/xsl"?>
<xsl:stylesheet
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns="http://www.w3.org/2000/svg"
        version="1.0">
```

```
<xsl:output method="xml" indent="yes" media-
                           type="image/svg+xml"/>

<xsl:template match="statistiche">
 <svg width="200" height="900">
    <!-- ridefinisco le coordinate dell'area di
                                     lavoro -->
   <g id="categorie" transform="translate(10,-
                                          30)">
     <!-- tramite XPATH ciclo su tutti i tag categorie
                                            -->
     <xsl:for-each select="categorie">
      <!-- inserisco gli elementi testuali e le forme
                                  geometriche -->
              <text x="1"
 y="{position()*45+30}" style="font-family:arial;">
                    <xsl:value-of
                         select="capo"/>
              </text>
              <xsl:variable
            name="vendite" select="vendite"/>
              <rect x="1"
         y="{position()*45+32}" height="25"
        width="{$vendite*3}" style="fill:red"
                        stroke="blue"/>
              <text x="1"
 y="{position()*45+50}" style="font-family:arial;">
                    <xsl:value-of
                     select="vendite"/>%
              </text>

     </xsl:for-each>

   </g>
 </svg>
</xsl:template>
```

Il risultato sarà il seguente:

20%	pantaloni
15%	sciarpe
15%	giubbotti
50%	scarpe

Naturalmente con SVG è possibile fare molto di più, ma questo semplice esempio è più che sufficiente per dare un assaggio della grande potenza e versatilità di XML.

Da notare il modo in cui vengono richiamati i vari tag: per selezionarli abbiamo infatti utilizzato le potenti funzionalità offerte dal linguaggio XPATH. Questo linguaggio ci consente di effettuare interrogazioni sul file XML, proprio come se stessimo eseguendo una query su un DB.
L'approfondimento delle tante funzionalità offerte da XSL è lasciato, a questo punto, alla buona volontà del lettore, che potrà anche beneficiare dei file allegati all'articolo: ora, infatti, è venuto il momento di interfacciare XSLT con Ajax, per sfruttare tutti i vantaggi e le potenzialità offerte dall'associazione di queste due formidabili tecniche di programmazione.

AJAX E XSLT: COPPIA VINCENTE

Con Ajax, come sappiamo bene, è possibile far interagire il nostro client con il server in modo completamente asincrono e trasparente agli occhi dell'utente. Ajax, inoltre, supporta perfettamente l'invio della risposta HTTP in formato XML. La domanda, a questo punto, può essere: possiamo creare un'applicazione che riassuma gli evidenti vantaggi di Ajax con quelli appena visti, di XSLT ?
La risposta, fortunatamente, è affermativa. Per realizzare un simile sincronismo possiamo lavorare nel seguente modo:

- la prima volta che ci si collega all'applicazione, viene effettuata una chiamata sincrona al server tramite Ajax, che ha lo scopo di scaricare in locale il file XSL necessario per operare le trasformazioni;

- successivamente, invece, il server si limita a generare la risposta XML contenente i dati da elaborare, mentre la creazione dinamica della pagina web sarà affidata al processore XSLT direttamente sul client.

A titolo di esempio riporto, in particolare, la funzione Javascript che invia la richiesta al server per scaricare il file XSL:

```
//carichiamo il file XSL
function carica_XSL()
{
  xml_impaginazione.open("GET", "griglia.xsl", false);
                         //il file viene scaricato in locale
  xml_impaginazione.send(null);
  if (this.DOMParser)
  {
    var parser_dom = new DOMParser();
    documento_xsl =
```

```
parser_dom.parseFromString(xml_impaginazione.res
                                ponseText, "text/xml");
 }
 else if (window.ActiveXObject) // per IE
 {
   documento_xsl = crea_Msxml2();  // istanzio il
                                              parser
   documento_xsl.async = false;   // il caricamento
      sul client del foglio XSL deve avvenire in modo
                                            sincrono
documento_xsl.load(xml_impaginazione.responseXML
                                                  );
 }
}
```

Da notare che il file rimane caricato in modo permanente, finché la cache non viene volutamente svuotata o non scade, a seconda delle impostazioni del browser. Qui di seguito è invece riportata la funzione che richiede al server la risposta XML per l'aggiornamento del client:

```
function carica_pagina(numero)
{
  if (xml_impaginazione) // se esiste
                              l'XMLHttpRequest object
  {
        if (xml_impaginazione.readyState == 4 ||
           xml_impaginazione.readyState == 0)
        {
        // se lo stato dell'oggetto non è occupato o
                           non è ancora stato utilizzato
                    var parametri =
           "impaginazione.php?pagina=" + numero;
                    xml_impaginazione.open("GET",
   parametri, true); // per caricare le varie pagine, la
      richiesta può tranquillamente essere asincrona
xml_impaginazione.onreadystatechange =
                                   gestisci_risposta;
                    xml_impaginazione.send(null);
        }
  }
}

...
```

Questo, infine, è il codice PHP che genera la risposta XML lato server. Da notare che, in questo caso, volendo essere sicuri che la risposta ritornata al client non venga memorizzata in cache (visto che vengono eseguite richieste GET sempre uguali), inviamo anche gli header giusti attraverso il codice che segue:

```
...
        if(ob_get_length()) ob_clean();
        header('Expires: Fri, 25 Dec 1980 00:00:00
                          GMT'); // epoca nel passato
             header('Last-Modified: ' . gmdate('D, d M Y
                                   H:i:s') . 'GMT');
             header('Cache-Control: no-cache, must-
                                         revalidate');
             header('Pragma: no-cache');
             // genero output XML
             header('Content-type: text/xml');
             $risposta = '<?xml version="1.0"
                                  encoding="UTF-8"?>'
                                        . '<info>'
                          . '<dati>'
                          . $stringa_xml
                          . '</dati><pagine>';

             for ($conta=1; $conta <= $num_pagine;
                                            $conta++)
             {
              if ($conta == $pagina)
              {
                    $risposta .= "<pagina><numero
            corrente='si'>$conta</numero></pagina>";
              }
              else
              {
                    $risposta .= "<pagina><numero
           corrente='no'>$conta</numero></pagina>";
              }
             }
             $risposta .= '</pagine></info>';
                    echo $risposta;
?>
```

Non si può, tuttavia, negare che tale soluzione soffra ancora di qualche inconveniente legato alla relativa "gioventù" delle tecnologie prese in considerazione: i principali problemi, infatti, sono dovuti alla mancanza del supporto ad Ajax e XSLT dei browser più vecchi, e a ad una certa complessità nell'attività di sviluppo se si vogliono creare applicazioni cross-browser dotate di funzionalità discretamente avanzate.

È però altrettanto importante sottolineare che, in un futuro ormai prossimo, l'utilizzo di tali strumenti diverrà addirittura indispensabile se si vorrà realizzare software in piena regola con gli standard del W3C.

Enrico Viale

REQUISITI PER L'UTILIZZO DEI FILE DI ESEMPIO

Sul CD allegato sono disponibili i file di esempio dell'articolo. Per far funzionare correttamente tali file, occorre disporre di un browser recente, che supporti XSLT. La cartella relativa ad Ajax e XSLT deve essere inserita, invece, in un web-server, e per funzionare necessita di PHP e Mysql (è necessario, prima di tutto, creare il db ed eseguire il file SQL allegato).

L'AUTORE

Enrico Viale è specializzato nello sviluppo di applicazioni sia web-oriented che desktop. Chi desidera contattarlo per chiarimenti riguardo all'articolo, o per qualsiasi altro motivo, può farlo all'indirizzo enrico.viale@gmail.com.

WEB 2.0? FACILE CON PROTOTYPE

PROTOTYPE È UN FRAMEWORK JAVASCRIPT CHE FACILITA ENORMEMENTE LO SVILUPPO DI APPLICAZIONI WEB. IN QUESTO ARTICOLO IMPAREREMO A SFRUTTARLO PER OTTENERE IL MEGLIO DALLE VOSTRE APPLICAZIONI BASATE SU AJAX

Prototype è un framework JavaScript che ha come scopo facilitare lo sviluppo di applicazioni web dinamiche. Questa è, più o meno, la traduzione in italiano della frase con cui Prototype si presenta al pubblico. Questa presentazione è, a mio avviso, un po' troppo modesta. In realtà Prototype, oltre a facilitare lo sviluppo, incrementa la produttività del programmatore in modo impressionante. A questo proposito facciamo subito un esempio. Chi utilizza Ajax per lo sviluppo di applicazioni web si sarà, molto probabilmente, trovato di fronte al seguente scenario. In pratica, avete una pagina web e volete aggiornare una parte di essa con il codice HTML ottenuto come response di una chiamata Ajax. Bene, per raggiungere questo risultato utilizzando Prototype vi basta scrivere qualcosa del genere:

```
new Ajax.Updater('container', './page.htm');
```

Quella singola riga di codice esegue una chiamata Ajax alla pagina page.htm e appena riceve il response ne inserisce il contenuto nell'elemento che ha come id container. Esso, il più delle volte, sarà un <div>, ma può tranquillamente essere qualsiasi altro elemento HTML identificato da un id.

Se invece non utilizzate Ajax ma solo JavaScript non preoccupatevi, Prototype fornisce un insieme di funzionalità che vi entusiasmeranno. Guardate, ad esempio, il seguente codice:

```
var prices = [10, 12, 25, 28, 50, 100];
// lowPrices varrà [10, 12, 25]
var lowPrices = prices.select( function(value)
                    {
                        return value <= 25;
                }
                );
```

In pratica, questo codice crea un array ed in seguito ne seleziona un suo sottoinsieme in base al valore restituito da una funzione che passiamo noi come parametro del metodo select. Se la cosa non è chiarissima niente timori. Per ora focalizzate l'attenzione solo nel risultato ottenuto con quel semplice codice. Nel prosieguo dell'articolo saranno analizzate in dettaglio questa e tante altre funzionalità di Prototype. Quello che vedremo vi consentirà di utilizzare Prototype per risolvere una buona parte dei problemi più ricorrenti. Per comprendere appieno tutte le funzionalità del framework vi consiglio di dare un'occhiata al codice sorgente di Prototype. È il miglior metodo per carpire tutti i segreti di questo meraviglioso strumento. Potete, inoltre, riferirvi alle API Docs che trovate sul sito ufficiale.

REQUISITI

Conoscenze richieste
Buone di JavaScript, Base di Ajax.

Software
Prototype, Scriptaculous, Rico

Impegno

Tempo di realizzazione

UN PRIMO ASSAGGIO

Prototype è un framework basato su JavaScript. Come tale esso facilita lo sviluppo di applicazioni web basate su JavaScript. È chiaro che tutto ciò che può essere fatto con Prototype, lo potete fare utilizzando JavaScript allo stato puro, ma perché complicarsi la vita? Ad esempio, è più veloce scrivere *document.getElementById ("myElement")* oppure *$("myElement")* per raggiungere lo stesso risultato? Se la risposta ricade sulla seconda opzione, allora vi consiglio di continuare con la lettura del presente articolo dato che $ è solo una delle fantastiche funzioni offerte da Prototype.

Abbiamo già visto la funzione $ che è un alias per *document.getElementById*. E' possibile passare anche più id a tale funzione. In tal caso il risultato sarà un array contenente gli elementi relativi agli id passati come parametri. Ecco un esempio:

```
<!-- HTML -->

<div id="alfa">Contenuto dell'elemento alfa</div>
<div id="beta">Contenuto dell'elemento beta</div>
// JavaScript

var elems = $("alfa", "beta");
```

```
alert(elems[1].innerHTML); //visualizza
                        "Contenuto dell'elemento beta"
```

Un'altra funzione molto utile è $F. Riceve in ingresso l'id di un elemento di un form e restituisce il valore di tale elemento. Ad esempio, il seguente codice visualizza la sigla della provincia selezionata tramite una combo box.

```
<!-- HTML -->
<select name="province" id="province">
 <option value="RC" selected="selected">Reggio
                                 Calabria</option>
 <option value="MI">Milano</option>
 <option value="CS">Cosenza</option>
 <option value="FI">Firenze</option>
</select>

// JavaScript
var sigla = $F("province");
alert(sigla); // visualizza "RC"
```

JAVASCRIPT ORIENTATO AGLI OGGETTI

Quando si parla di JavaScript, spesso, gli si attribuisce la colpa di essere un linguaggio poco elegante e "disordinato". Col passare del tempo, tuttavia, ci si rende conto che la scrittura di codice elegante dipende, in gran parte, dal programmatore. Utilizzando Prototype è possibile scrivere facilmente codice JavaScript orientato agli oggetti, con tutti i vantaggi che ne derivano. Ad esempio, ecco come è possibile definire una classe e crearne un'istanza:

```
// Definizione della classe
var Book = Class.create();
// Metodi
Book.prototype =
{
        // Costruttore
        initialize : function(author, title, editor, isbn,
                                                  price)
        {
                this.author = author;
                this.title = title;
                this.editor = editor;
                this.isbn = isbn;
                this.price = price;
        },
// Un metodo
        getDiscountedPrice : function(discount)
        {
                return this.price - ((this.price *
                                   discount) / 100);
        },
```

```
// Un altro metodo
        alertAuthor : function()
        {
                alert(this.author);
        },

        // Ancora un altro metodo
        toString : function()
        {
                return "Autore: " + this.author +
                                            "\n" +
                          "Titolo: " + this.title + "
"Editore: " + this.editor + "\n" +
"ISBN: " + this.isbn + "\n" +
"Prezzo: " + this.price;
        }
};

// Crea un oggetto di tipo Book
var book = new Book("Mario Rossi",
                            "Un titolo",
                            "L'editore",
                            "123456789",
                            30);
// Visualizza il libro e il suo prezzo scontato
alert(book.toString() + "\n" + "Prezzo scontato del
            25%: " + book.getDiscountedPrice(25));
```

Come potete vedere, per definire una classe, usando Prototype, basta utilizzare la sintassi:

```
var NomeClasse = Class.create();
```

Aggiungere dei metodi alla classe è altrettanto semplice:

```
NomeClasse.prototype =
{

        // Metodi
        [...]
};
```

Notate che è possibile definire un costruttore. Vi basta dichiarare un metodo il cui nome è, per convenzione, **initialize** e Prototype richiamerà tale metodo ogni qual volta si creerà un'istanza della classe in questione. Ovviamente, oltre al costruttore, potete definire altri metodi. Nel nostro esempio abbiamo creato i metodi **getDiscountedPrice**, **alertAuthor** e ridefinito **toString**. Il primo restituisce il prezzo del libro dopo aver applicato lo sconto passato come parametro al metodo. Il secondo, invece, visualizza l'autore del libro. Il metodo toString restituisce una rappresentazione in stringa della classe. Creare, poi, un oggetto di una classe è simile ad altri linguaggi:

```
var book = new Book("Mario Rossi",
                    "Un titolo",
                    "L'editore",
                    "123456789",
                    30);
```

Così facendo, ad esempio, abbiamo creato un oggetto di tipo Book. Richiamare i metodi sull'oggetto creato è altrettanto semplice:

```
alert(book.toString() + "\n" + "Prezzo scontato del
        25%: " + book.getDiscountedPrice(25));
```

Il risultato del codice precedente è illustrato in figura 1.

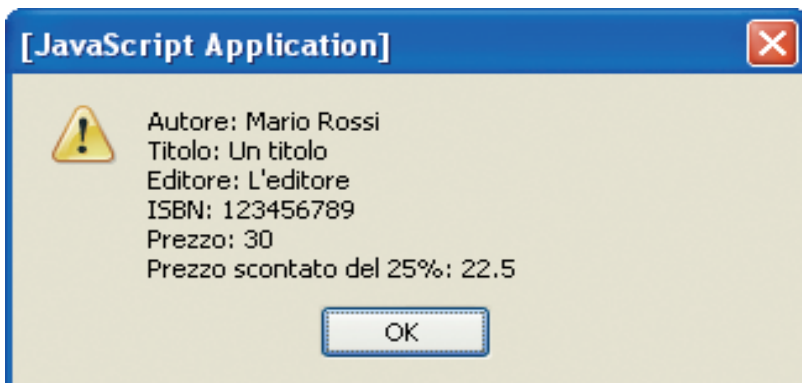



Fig. 1: alert contenente la descrizione del libro ed il prezzo scontato.

Prototype consente anche di estendere una classe. Ciò può essere fatto grazie al metodo extend di Object. Questa è la dichiarazione di extend estratta direttamente dal codice sorgente di Prototype:

```
Object.extend = function(destination, source) {
  for (var property in source) {
    destination[property] = source[property];
  }

  return destination;
}
```

In pratica, extend copia tutte le proprietà ed i metodi di un oggetto in un altro. Il parametro destination è l'oggetto che sarà esteso con le proprietà ed i metodi di source.

SFRUTTARE I FRAMEWORK

Prototype è un framework. I framework sono progettati con lo scopo di facilitare lo sviluppo di software. Essi permettono agli sviluppatori di spendere più tempo a risolvere problemi legati alla soddisfazione dei requisiti che non a trattare con dettagli di basso livello che non hanno nulla a che vedere con i requisiti.

A VOLTE "THIS" FA BRUTTI SCHERZI

Forse non tutti sanno che, in JavaScript, una funzione è un oggetto vero e proprio. Prototype estende la classe Function, di cui ogni funzione è un'istanza, aggiungendo due metodi utilissimi: **bind** e **bindAsEventListener**. Questi due metodi trovano impiego, soprattutto, quando si stanno impostando i gestori degli eventi. Tuttavia, visto il titolo del paragrafo, vi starete chiedendo: "ma cosa ha a che fare tutto ciò con la parola chiave this?". Lo capirete subito dopo avere analizzato il seguente esempio (Book è la classe definita nell'esempio precedente):

```
<!-- HTML -->
<input type="button" id="elem" name="elem"
                  value="Visualizza autore"/>

// JavaScript
var book = new Book("Mario Rossi",
                    "Un titolo",
                    "L'editore",
                    "123456789",
                    30);
$("elem").onclick = book.alertAuthor;
```

Osservando il precedente codice viene spontaneo pensare che, non appena l'utente clicca sul bottone "Visualizza autore", viene mostrato un alert con il nome dell'autore. Tuttavia, se provate ad eseguire questo codice noterete che sarà visualizzato undefined. L'ultima riga di codice, infatti, non causa la semplice invocazione del metodo **alertAuthor**, quando si verifica l'evento click sul bottone. Quello che avviene dietro le quinte è un po' più complicato. Descrivendo a parole ciò che succede, quando scrivete qualcosa del tipo *$("elem").onclick = book.alertAuthor* abbiamo: "Sostituisci il corpo del metodo *onclick* del bottone con il corpo del metodo *alertAuthor* della classe Book". Per chiarire le cose riportiamo di seguito il metodo alertAuthor di *Book*:

```
alertAuthor : function()
{
        alert(this.author);
}
```

Quindi il corpo della funzione *onclick* del bottone ora è *alert(this.author)*. La parola chiave *this*, quindi, non si riferisce più alla classe *Book* ma al bottone vero e proprio. Il problema è che il button non ha nessuna proprietà di nome *author!* Per tale motivo viene visualizzato *undefined*. Questo è un problema molto ricorrente in cui s'imbattono, prima o poi, tutti i programmatori JavaScript. Attraverso Prototype è possibile risol-

vere tale problema con i metodi bind e bindAsEventListener di Function. Ad esempio, basta sostituire il codice precedente con quello che segue per risolvere la questione:

```
$("elem").onclick = book.alertAuthor.bind(book);
```

Adesso la parola chiave this continua a riferirsi all'oggetto book dato che è stata "bindata", ossia legata, ad esso. Traducendo in parole la precedente riga di codice abbiamo più o meno: "Sostituisci il corpo del metodo *onclick* del bottone con quello di alertAuthor e lega l'oggetto book alla parola chiave *this*".
Il metodo *bindAsEventListener* svolge lo stesso lavoro di *bind* ed in più passa un parametro alla funzione da chiamare. Tale parametro è l'oggetto *event* ed è passato in un modo che sia cross-browser. Il tutto in modo completamente trasparente per il programmatore.

ESTENSIONI DELLA CLASSE ARRAY

Prototype definisce un oggetto, *Enumerable*, il quale possiede un numero consistente di metodi utili per ciclare sugli array. Questo è possibile grazie al fatto che la classe Array, nativa di JavaScript, è stata estesa, attraverso *Object.extend*, con *Enumerable*. Facciamo subito qualche esempio.
Quello che segue è un esempio che illustra il metodo classico che utilizziamo per ciclare su un array in JavaScript:

```
var regioni = ['Calabria', 'Lombardia', 'Lazio'];

for(var i = 0; i < regioni.length; i++)
{
        alert(regioni[i]);
}
```

Fin qui nulla di nuovo. Vediamo, ora, come potremmo raggiungere lo stesso risultato in Prototype utilizzando il metodo *each*:

```
var regioni = ['Calabria', 'Lombardia', 'Lazio'];

regioni.each( function(value, index)
{

alert(value);
}
);
```

Il metodo each, come tanti altri di *Enumerable*, prende come argomento una funzione a cui ci si riferisce spesso con il nome di *iterator*. Tale funzione prende due parametri. Il primo, value, è l'elemento corrente dell'iterazione. In pratica svolge lo stesso ruolo di regioni[i] dell'esempio precedente. Il secondo, index, è l'indice corrente, ossia la variabile i di prima. Il parametro index è opzionale, ossia è possibile passare, come iteratore, una funzione che lavora solo su value. L'esempio seguente è, funzionalmente, analogo al precedente:

```
var regioni = ['Calabria', 'Lombardia', 'Lazio'];
regioni.each( function(value)
{
alert(value);
}
);
```

Ora, però, siamo onesti! Il metodo *each* non ha un'utilità pari alle altre funzionalità viste fino ad ora. Tuttavia, *Enumerable* fornisce tanti altri metodi che troverete sicuramente molto interessanti. Vi sono, ad esempio, vari metodi che possono essere usati per "filtrare" gli elementi di un array. Uno di questi l'abbiamo visto all'inizio dell'articolo. Lo riportiamo per comodità:

```
var prices = [10, 12, 25, 28, 50, 100];

// lowPrices varrà [10, 12, 25]
var lowPrices = prices.select( function(value)
                        {
                        return value <= 25;
                  }
            );
```

Il metodo select prende anch'esso, come each, un iterator come parametro. Questo metodo restituisce un array formato dagli elementi per i quali la funzione iteratore restituisce *true*.
Se avete la necessità di fare ricerche più avanzate vi è anche la possibilità di utilizzare le espressioni regolari (regex). Guardate, ad esempio, il seguente codice:

```
var regioni = ['Calabria', 'Lombardia', 'Lazio'];
// Visualizza Calabria e Lombardia
regioni.grep(/.*ia/, function(value)
                {
                  alert(value);
                }
                              );
```

Il metodo *grep*, come avete notato, prende due parametri. Il primo è una regex, mentre il secondo è il solito iterator. L'esempio precedente significa: "Selezionami gli elementi dell'array che iniziano con qualunque carattere e finisco-

no per ia". Il risultato, quindi, è la visualizzazione di due alert consecutivi che contengono Calabria e Lombardia. Non entrerò nel merito delle espressioni regolari in JavaScript dato che non è questo il contesto. Vi sono tanti altri metodi di Enumerable che varrebbe la pena esaminare e che non facciamo per ragioni di spazio. Vi consiglio, in ogni modo, di dare un'occhiata alle API Docs che trovate sul sito ufficiale.

ALTRE POTENTI FUNZIONALITÀ

Come ho già detto, Prototype è abbastanza grande come framework. Non sarebbe possibile esaminarlo in toto in uno o due articoli. Quelle che fornirò, ora, sono solo alcune delle feature esposte da questo potente framework.
Un oggetto molto interessante è Element. Ad esempio il suo metodo toggle può essere utilizzato per far apparire e nascondere un elemento HTML in modo alternativo. Facciamo un esempio per chiarire:

```
<input type="button" value="Cliccami"

    onclick="javascript:Element.toggle('toggable')"/>
<div id="toggable">

        Mi puoi far apparire e sparire cliccando il
                                    bottone sopra
</div>
```

Non appena l'utente clicca sul bottone Cliccami il contenuto del <div> viene nascosto e mostrato alternativamente. Altri due importanti metodi di Element sono show e hide. Possono essere usati, rispettivamente, per mostrare e nascondere un elemento.
Un altro oggetto di sicuro interesse è Event. Come avrete intuito dal nome, esso ha a che fare con gli eventi e la relativa gestione. Ovviamente, Event gestisce gli eventi in un modo compatibile con la stragrande maggioranza dei browser in circolazione senza che lo sviluppatore si preoccupi di ciò.
Ad esempio, è possibile utilizzare il metodo observe per "osservare" il verificarsi di un evento su un elemento. Ecco un esempio:

```
<!-- HTML -->
<div id="elem">
  Clicca qui
</div>
//Javascript

Event.observe($("elem"), "click", displayAlert, false);
function displayAlert()
{
          alert("ioProgrammo");
}
```

Non appena l'utente clicca su "Clicca qui" viene visualizzato un alert con la stringa ioProgrammo. In pratica, il metodo observe prende quattro parametri:

1. L'elemento da tenere sott'occhio.
2. Il tipo di evento: click, blur, keyup e così via.
3. La funzione di callback da chiamare al verificarsi dell'evento.
4. L'ultimo parametro indica se utilizzare la fase di capturing.

Per smettere di osservare un evento si può usare stopObserving:

```
Event.stopObserving($("elem"), "click", displayAlert,
                                            false);
```

Arrivati a questo punto ci rimane da analizzare la parte forse più interessante e potente dell'intero framework, ossia le API offerte per lavorare agevolmente con Ajax.

AJAX: UN GIOCO DA RAGAZZI CON PROTOTYPE

Prototype permette anche ai non esperti di Ajax di far uso di questa rivoluzionaria tecnologia. Uno dei problemi di Ajax, come tutte le tecnologie che hanno alla base JavaScript, è che bisogna tenere presente la differenza tra i browser in circolazione. Se, ad esempio, volete creare un'istanza dell'oggetto XMLHttpRequest, dovete utilizzare due strade diverse per le versioni di Internet Explorer precedenti la 7 e Firefox. Per le prime potreste usare qualcosa del tipo:

```
var xhr = new ActiveXObject("Microsoft.XMLHttp");
```

CAPTURING E BUBBLING

In JavaScript vi sono due modelli per la gestione di un evento: capturing e bubbling. Microsoft IE utilizza il bubbling, mentre buona parte degli altri browser, seguendo il modello di Netscape, utilizzano il capturing. Nella fase di capturing l'evento si propaga dall'elemento più esterno fino a quello in cui si è verificato l'evento stesso. Nella fase di bubbling l'evento si propaga in modo opposto, ossia dall'elemento in cui si è verificato a quello più esterno. In entrambe le fasi, la propagazione passa per tutti gli elementi intermedi tra il più esterno e quello in cui si verifica l'evento.

Per Firefox, IE 7, ed altri browser, invece, il codice da utilizzare è il seguente:

```
var xhr = new XMLHttpRequest();
```

Già questo ci fa capire che, come di consueto, le differenze tra i browser ci possono provocare grossi grattacapi. Prototype ci viene incontro per risolvere questa e tante altre differenze di implementazione tra i diversi browser. Tornando ad Ajax vediamo, ora, quali sono gli strumenti principali forniti da Prototype. Abbiamo visto, all'inizio dell'articolo, che questo potente framework ci consente di raggiungere risultati impressionanti scrivendo poche righe di codice che richiederebbero, invece, una grossa mole di lavoro lavorando direttamente con JavaScript. L'esempio fatto all'inizio è il seguente:

```
new Ajax.Updater('container', './page.htm');
```

Come abbiamo già detto, tale codice recupera, in modo asincrono, il contenuto della pagina page.htm e lo visualizza all'interno dell'elemento *container*, il quale, il più delle volte è un <div> del tipo:

```
<div id="container">
</div>
```

Prototype consente un'ampia personalizzazione. Ad esempio, sarebbe possibile definire una funzione di callback da chiamare in caso di fallimento della chiamate Ajax e, se vogliamo, specificare il metodo HTTP da utilizzare per la chiamata stessa:

```
// JavaScript
new Ajax.Updater(
        "container",
        "page.htm",
        {
            method : "get",
          onFailure : function() {
                    alert("Si è verificato un
                                     errore");
                    }
        }
);

<!-- HTML -->
<div id="container">
</div>
```

Il terzo parametro di Ajax.Updater è un oggetto, in questo caso espresso in modo letterale. È possibile specificare diverse proprietà per quest'oggetto in modo da personalizzare al massimo la richiesta. Nell'esempio precedente abbiamo visto *method* e *onFailure*. La prima specifica il metodo HTTP da usare, mentre la seconda indica la funzione da chiamare in caso di fallimento. Alcuni di voi si staranno chiedendo: "E se invece non devo chiamare una pagina HTML statica, ma piuttosto un componente server, come una pagina PHP, che mi restituisce codice HTML a seconda dei parametri che riceve?". Prototype ha, ovviamente, una risposta anche a questo:

```
// JavaScript
new Ajax.Updater(
        "container",
        "component.php",
        {
            method : "post",
            parameters :
"bookId=5&language=it",
          onFailure : function() {
                    alert("Si è verificato un
                                     errore");
                    }
        }
);
<!-- HTML -->
<div id="container">
</div>
```

Nell'esempio precedente abbiamo chiamato un'ipotetica pagina PHP, component.php, utilizzando il metodo POST e passandogli i parametri *bookId* e *language*. Da notare che i parametri sono passati nella forma di query string, ossia *key1=value1&key2=value2* e così via.
Ovviamente, component.php utilizzerebbe tali parametri opportunamente per generare il codice HTML necessario, ad esempio, a visualizzare la descrizione in italiano del libro identificato da quell'ID.
Come vedete vi sono numerose possibilità di customizzazione per quanto riguarda la richiesta. Chiaramente non le possiamo esaminare tutte, anche se quelle appena viste sono le più utilizzate.
Se, tuttavia, il vostro componente lato server non ritorna codice HTML ma qualche altro formato come XML o JSON niente paura, Prototype vi fornisce una classe che fa al caso vostro. Stiamo parlando di *Ajax.Request*; facciamo subito un esempio:

```
new Ajax.Request(
                    "component.php",
                    {
                              onSuccess :
                          processResponse,
                              onFailure :
```

```
                                notifyFailure,
                                parameters :
                                "key1=value1&key2=value2"
                                }
);
function processResponse(xhr)
{
        alert("Il response ricevuto è: " +
                                xhr.responseText);
}
function notifyFailure()
{
        alert("Si è verificato un errore");
}
```

Il codice precedente esegue una chiamata asincrona alla pagina component.php specificando, tramite l'oggetto letterale:

- La funzione da chiamare in caso di successo. Tale funzione è specificata tramite la proprietà *onSuccess*.
- La funzione da chiamare in caso di fallimento, indicata tramite la proprietà *onFailure*.
- I parametri da utilizzare, specificati tramite la proprietà parameters.

La funzione *processResponse*, invocata in caso di successo, riceve in ingresso l'oggetto XMLHttpRequest vero e proprio che è stato utilizzato internamente da Prototype per la chiamata Ajax. Tale oggetto espone la proprietà *responseText* che contiene il responso del server, qualsiasi esso sia.
Esaminiamo, ora, un altro caso comune. Vi sarà capitato, visitando un sito che impiega Ajax per le chiamate al server, che vi venga presentata una GIF atta a notificare il "work in progress" del sistema. Un esempio può essere una clessidra. A tal proposito potrebbe essere utile avere la possibilità di specificare un <div> della pagina, contente la GIF in questione, da mostrare ogni volta che parte una chiamata Ajax e nascondere non appena la chiamata ritorna. Prototype ha pensato anche a questo:

```
// JavaScript
Ajax.Responders.register(
                {
                        onCreate :
                                displayLoading,
                        onComplete :
                                hideLoading
                }
);
function displayLoading()
{
        Element.show("loading");
}

function hideLoading ()
{
        Element.hide("loading");
}

<!-- HTML -->
<div id="loading" sytle="visibility: hidden;">
        <img src="loading.gif" />
</div>
```

Il codice precedente invoca la funzione *displayLoading* non appena parte una chiamata Ajax e *hideLoading* quando la chiamata termina. Le due funzioni utilizzano i metodi *show* e *hide* di Element, rispettivamente per mostrare e nascondere la GIF contenuta all'interno del <div> loading.

QUALCHE PAROLA SU AJAX

Ajax sta per Asynchronous JavaScript + XML. Il termine è stato coniato da Jesse James Garrett, presidente di Adaptive Path, in un articolo che potete trovare al seguente link: http://www.adaptivepath.com/publications/essays/archives/ 000385.php
In sostanza, Ajax è un insieme di tecnologie utilizzate congiuntamente per sviluppare applicazioni, cosiddette, Web 2.0. Di questo insieme di tecnologie fanno parte:

1. **(X)HTML e CSS, utilizzati per lo strato di presentazione.**
2. **DOM, per interagire dinamicamente col documento.**
3. **XML, come formato d'interscambio dati.**
4. **XMLHttpRequest, per il recupero asincrono dei dati.**
5. **JavaScript, come "collante" delle precedenti.**

Vi è da dire, tuttavia, che non è obbligatorio utilizzare XML come formato di interscambio dati.
In alcuni contesti è preferibile usare JSON o qualche altro formato customizzato.

PROTOTYPE COME FONDAMENTA PER LIBRERIE DI ALTO LIVELLO

A riprova della straordinaria potenza di Prototype vi sono diverse librerie grafiche costruite su di esso. Tra le più utilizzate abbiamo: Scriptaculous e Rico. Offriremo solo un piccolo assaggio di entrambe le librerie dato che, vista la loro dimensione, non basterebbe l'intera rivista per analizzarle entrambe in dettaglio.
Scriptaculous offre una vasta libreria per lo sviluppo di accattivanti interfacce grafiche (UI) basate su JavaScript. Vi sono componenti per l'animazione, il drag&drop, controlli Ajax ed oggetti per la manipolazione del DOM. Ecco, ad esempio, alcuni effetti di animazione:

```
<html>
<head>
  <!-- Includiamo i file che ci servono -->
  <script src="./scripts/prototype.js"
                        type="text/javascript"></script>
  <script src="./scripts/scriptaculous.js"
                        type="text/javascript"></script>
</head>
<body>
<!-- Effetto Shake -->
<input type="button"
                                  value="Clicca
                        per vedere l'effetto Shake"
onclick="new Effect.Shake($('element'))" />
<br /><br />
<!-- Effetto Puff -->
<input type="button"
                                  value="Clicca
                         per vedere l'effetto Puff"
onclick="new Effect.Puff($('element'))" />
<br /><br />
<!-- Elemento che subirà gli effetti -->
<div id="element" style="width: 200px; border: 1px
                                      solid red;">
           Questo è l'elemento che subirà gli effetti
                               grafici di cui sopra
</div>
[...]
```

Il codice precedente illustra solo due delle animazioni fornite da Scriptaculous. Cliccando sul primo bottone si vedrà il <div> element "shakerare" un po'. Il secondo, invece, farà sparire il <div> in un effetto di "puff". Provate il codice precedente per rendervi conto visivamente di ciò che succede. Abbiamo raggiunto quei particolari effetti grafici utilizzando una sola riga di codice. La sintassi generale, per gli effetti, è la seguente:

```
new Effect.EffectName(element);
```

Dove EffectName è il nome dell'effetto da utilizzare, come Puff, BlindDown, Fade ecc. Il parametro element è l'elemento su cui applicare l'effetto. Ovviamente, vi è la possibilità di customizzare gli effetti a proprio piacimento. Guardate la documentazione presente sul sito ufficiale per ulteriori informazioni. Un'altra straordinaria libreria per lo sviluppo di UI è Rico. A parte Ajax, drag&drop ecc. Rico offre effetti cinematici molto interessanti. Vi è mai capitato, ad esempio, di voler racchiudere qualcosa all'interno di un rettangolo con gli angoli arrotondati? Rico vi permette di farlo con una quantità di codice irrisoria:

```
<html>
<head>
    <!-- Includiamo i file che ci servono -->
```

```
            <script src="./scripts/prototype.js"
                        type="text/javascript"></script>
            <script src="./scripts/rico.js"
                        type="text/javascript"></script>

</head>
<body>
<input type="button"
                                      value="Clicca
                per arrotondare il <div> sottostante"
onclick="Rico.Corner.round($('element'))" />
<br /><br />
<div id="element" style="background-color:
             #0000ff; color: #ffffff; width: 200px;">

            Se clicchi il bottone sopra i miei angoli
                                 saranno arrotondati
</div>
[...]
```

Il codice appena visto visualizza un bottone ed un box con sfondo blu e scritta bianca. Non appena l'utente clicca sul bottone gli angoli del box, rappresentato dal <div>, saranno arrotondati. Il risultato è visibile in **figura 2**.

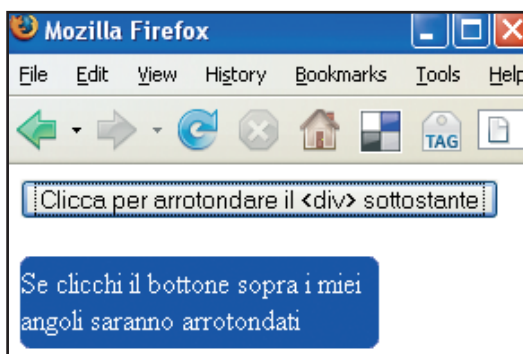



***Fig. 2:** Effetto angoli arrotondati ottenuto usando Rico.Corner.round.*

Ovviamente anche con Rico è possibile personalizzare gli effetti. Riferitevi al sito ufficiale per ulteriori informazioni ed esempi.

CONCLUSIONI

In questo articolo avete avuto modo di assaggiare la potenza di Prototype. Ovviamente, quelle viste sono solo alcune delle potenzialità offerte dal framework. Esso è così ben fatto che altre librerie di alto livello, come Scriptaculous e Rico, lo usano come base del loro codice. Con tutti questi strumenti a nostra disposizione solo la fantasia ci separa dal costruire applicazioni web di ultima generazione.

Alessandro Lacava

INTRODUZIONE AD ASP.NET AJAX

ARRIVA FINALMENTE IN VERSIONE UFFICIALE IL FRAMEWORK DI MICROSOFT DEDICATO AL WEB 2.0. IMPARIAMO COME SFRUTTARLO A FONDO PER OTTENERE APPLICAZIONI WEB CON INTERFACCE COMODE COME QUELLE LATO DESKTOP

Tutti (o quasi) lo conoscono, molti già lo utilizzano. Ajax è uno dei pilastri del WEB 2.0-Una delle tecnologie "alla moda" degli ultimi tempi e le sue incarnazioni sono molteplici, non ultima quella di cui ci occuperemo in queste righe, ovvero quella che va sotto il nome di ASP.NET Ajax, che altro non è che l'implementazione per gli sviluppatori .NET (ma non solo) di questa tecnologia da parte di Microsoft.
In questo articolo quindi non parleremo di cosa sia e come funzioni Ajax (di cui in modo generale si è parlato già in numerosi articoli nei numeri precedenti e, per un veloce approfondimento, si rimanda all'apposito box presente su queste stesse pagine), ma piuttosto faremo conoscenza del framework creato da Microsoft per rendere lo sviluppo con questa tecnologia ancora più facile e trasparente agli sviluppatori ASP.NET e non solo.
Per toccare subito con mano le funzionalità ed i nuovi strumenti, verrà utilizzato un piccolo progetto di esempio che consiste in un portale per la pubblicazione di articoli, una sorta di mini-CMS (Content Management System) molto semplice costruito al solo scopo di avere un caso d'uso reale su cui testare le caratteristiche di ASP.NET Ajax. In particolare il sito avrà un modulo per la realizzazione di sondaggi, ed un altro modulo per la visualizzazione di articoli, come mostrato in **Figura 1**. Entrambi sono implementati come User Control ASP.NET.
Specifichiamo subito che il codice di esempio ha il solo scopo di evidenziare e mostrare in modo

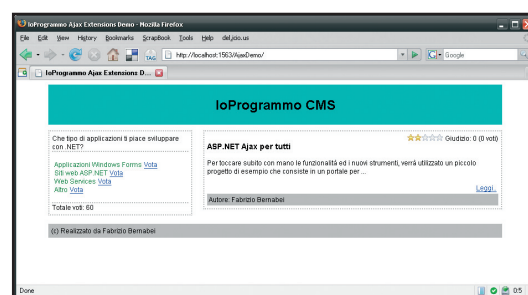



Fig. 1: L'applicazione di esempio in esecuzione

introduttivo l'utilizzo dei nuovi strumenti, quindi le funzionalità saranno in realtà solamente simulate (per esempio i contenuti non saranno effettivamente caricati da una fonte dati) e nel mostrare il codice ci si limiterà solamente alla parte che coinvolge Ajax, rimandando per un approfondimento o per semplice curiosità al progetto completo allegato all'articolo.
Ma iniziamo per prima cosa esplorando cosa viene messo a disposizione dal framework, che si compone delle seguenti parti:
ASP.NET Ajax Extensions: rappresenta il cuore vero è proprio della libreria; consta di un insieme di classi JavaScript (riutilizzabili anche in altre piattaforme diverse da ASP.NET) ed una serie di controlli server di base per l'integrazione in ASP.NET.
ASP.NET Ajax Futures: controlli server precedentemente inclusi nel core, fino all'ultima CTP "Atlas" (vedi box "Da Atlas ad ASP.NET Ajax Extensions"). Attualmente passati "in secondo piano" e distribuiti sottoforma di anteprima e di cui è prevista l'inclusione nel core nelle versioni future di ASP.NET Ajax.
ASP.NET Ajax Control Toolkit: libreria aggiuntiva di server controls già pronti dotati di funzionalità Ajax utilizzabili nelle applicazioni ASP.NET.
Ma iniziamo subito a sperimentare cosa è possibile fare con i nuovi strumenti messi a disposizione. Se non fosse ancora stato fatto, è necessario installare le estensioni Ajax. Per farlo basterà lanciare il file di setup scaricabile dal sito ufficiale http://ajax.asp.net che si occuperà di copiare le dll

COME INIZIARE

Per la realizzazione del progetto discusso in questo articolo è necessario avere installato il framework .NET 2.0. Per la scrittura del codice è consigliato (anche se non obbligatorio) utilizzare Visual Studio anche nella versione gratuita (Express). Inoltre è necessario installare ASP.NET Ajax Extensions ed ASP.NET Ajax Control Toolkit, entrambi da poco rilasciati in versione definitiva 1.0 e liberamente scaricabili dal sito ufficiale http://ajax.asp.net.

necessarie nel PC, registrare l'assembly "core" nella GAC e creare dei template di progetti per Visual Studio 2005.
Fatto questo è il momento di creare un nuovo "Web Site" in Visual Studio usando come template "ASP.NET Ajax-Enabled Web Site", come mostrato in **Figura 2**. In realtà potremmo creare un qualsiasi

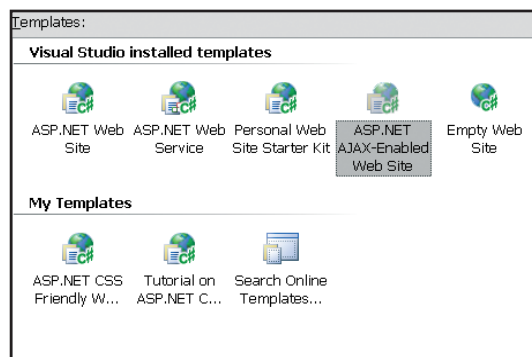



Fig. 2: Il template per la creazione di un nuovo sito web con supporto ad Ajax

altro tipo di progetto web (o aggiungere il supporto Ajax ad un sito già sviluppato), ma usando il template installato verranno automaticamente creati i riferimenti necessari alle librerie nel progetto ed il web.config modificato con le aggiunte necessarie per l'utilizzo di Ajax anche durante lo sviluppo con il supporto dell'Intellisense.
Nel progetto, sono presenti due User Control (come anticipato non ci soffermeremo sull'implementazione vera e propria, il cui codice è comunque presente nel codice allegato) ai quali verranno aggiunte funzionalità Ajax.
Il primo controllo implementa un modulo per effettuare sondaggi ed inizialmente funziona in modo "classico" leggendo ed aggiornando dati da un file xml per mezzo di postback. Il controllo dichiarato nel sorgente della pagina default.aspx è il seguente:

```
<uc1:BoxSondaggio ID="BoxSondaggio1"
                              runat="server" />
```

Internamente il controllo si occupa di generare a runtime dei pulsanti per ogni voce del sondaggio letta da un file xml esterno. Ad ogni pressione di uno dei pulsanti viene generato un postback per calcolare, nel codice eseguito dalla classe server, i valori aggiornati per le voci del sondaggio e quindi per aggiornare visivamente i valori del sondaggio stesso, come mostrato in **Figura 3**.
Il limite di questo approccio è dato dal fatto che ad ogni azione di postback, l'intera pagine deve essere richiesta al server e ri-visualizzata dal browser client, quando invece basterebbe "ridisegnare" solamente l'area ed i dati interessati dall'azione richiesta. Nel caso specifico non ha senso che il ser-

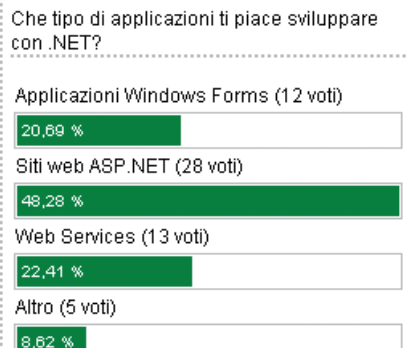



Fig. 3: Il risultato di una votazione

ver rimandi al client i dati dell'articolo presenti nel secondo controllo ed altri eventuali dati della pagina, ma basterebbe che sul server fossero aggiornati i dati del sondaggio, e solo questi restituiti al client per l'aggiornamento della visualizzazione della singola porzione della pagina interessata.
Per ottenere questo comportamento (di cui potremmo aver bisogno anche su un progetto web già in funzione) basta utilizzare uno dei controlli server fondamentali messi a disposizione da ASP.NET Ajax Extension, l'UpdatePanel. Questo controllo non è altro che un contenitore al cui interno possiamo posizionare i controlli server che vogliamo si comportino come descritto in precedenza. Tutto quello che dobbiamo fare è modificare il codice della pagina default.aspx in questo modo

```
<asp:ScriptManager ID="ScriptManager1"
                              runat="server" />
...
<asp:UpdatePanel ID="PnlVotazione" runat="server">
  <ContentTemplate>
    <uc1:BoxSondaggio ID="BoxSondaggio1"
                              runat="server" />
  </ContentTemplate>
</asp:UpdatePanel>
```

Dal codice sono subito evidenti due cose: la prima che, il controllo per la gestione dei sondaggi preesistente, è ora racchiuso in un controllo *UpdatePanel* che gli consentirà in fase di postback di aggiornare solamente il suo contenuto invece che l'intera pagina, senza scrivere una riga di codice nel controllo vero e proprio (che anzi neanche si "accorge" dell'esistenza del nuovo contenitore). La seconda è la presenza di un'altra riga di codice che definisce un altro controllo *ScriptManager*, questo è fondamentale per il funzionamento di Ajax in quanto ha il compito di generare nell'html finale, le inclusioni del codice javascript necessario per le funzionalità utilizzate dai controlli. Ovviamente questo controllo deve essere presente una sola volta nella pagina anche se questa contiene più controlli Ajax (se usiamo la ToolBox visibile in **Figura 4** per aggiungere controlli

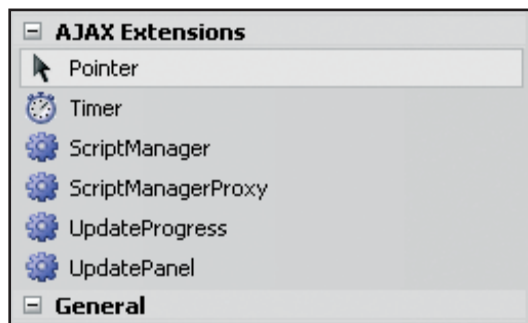



Fig. 4: La toolbox dei nuovi controlli Ajax

Ajax trascinandoli sulla pagina, lo *ScriptManager* verrà aggiunto automaticamente). Con questa semplice aggiunta abbiamo ottenuto il risultato voluto, ovvero facendo clic su uno dei link per la votazione del sondaggio, viene scatenato un postback con la differenza però che la pagina non viene richiesta completamente, ma viene fatta una richiesta "per mezzo di Ajax" ottenendo come risultato il solo codice html per il controllo contenuto nell'Update Panel (vedi **Figura 3**) e il non ricaricamento completo della pagina, con tutti i benefici per l'utente che può utilizzare una interfaccia più reattiva ed amichevole. Questo utilizzo di "base" dell'*UpdatePanel* porta vantaggi immediati fornendo la possibilità di aggiungere funzionalità Ajax con pochissimo sforzo anche ad applicazioni web preesistenti senza dover modificare assolutamente nulla della logica applicativa precedente. Allo stesso modo ci sono altri controlli e comportamenti utili che possono essere utilizzati ed aggiunti ad applicazioni già completate. Ad esempio in un contesto del genere sarebbe utile dare un feedback dell'attività all'utente durante l'invio del voto. Grazie ad un altro controllo presente in Ajax Extensions questo problema è facilmente risolvibile; basterà aggiungere un altro controllo Ajax alla pagina (nell'esempio è stato inserito direttamente all'interno dell'*UpdatePanel*) del tipo *UpdateProgress* come segue

COS'È AJAX E IL WEB 2.0

Il termine Ajax (acronimo di Asynchronous JavaScript And XML) in realtà rappresenta l'unione di varie tecnologie, tra l'altro neanche recentissime, utilizzate allo scopo di alleggerire i dati scambiati tra client e server in una applicazione web con lo scopo di renderne l'utilizzo più "ricco", cioè responsivo e dotato di comportamenti propri delle applicazioni non web.
Il Web 2.0 invece altro non è che un nuovo modo di concepire siti web (nella maggior parte dei casi usati per siti di "aggregazione" e scambio di opinioni/dati/risorse fra utenti, identificabile anche come social-web), in cui si fa largo uso di caratteristiche Ajax e grafica ricercata ma minimalista per ottenere maggiore interattività e velocità. Fra i primi e più famosi si possono citare http://del.icio.us per la memorizzazione e la condivisione di bookmarks e http://flickr.com per la catalogazione e condivisione di fotografie digitali, che sono solo alcuni dei siti web di "nuova generazione".

```
...
<asp:UpdateProgress ID="BoxAttesa"
        AssociatedUpdatePanelID="PnlVotazione"
                runat="server">
        <ProgressTemplate>
            Aggiornamento dati votazione...
        </ProgressTemplate>
</asp:UpdateProgress>
</ContentTemplate>
```

Questo controllo se collegato ad un *Updatepanel* per mezzo della proprietà *AssociatedUpdate PanelID*, rimane non visibile finché all'interno dell'*UpdatePanel* associato non si verifica un postback, nel qual caso diventa visibile fino a tornare a nascondersi al termine dell'evento di postback. Con il semplice codice inserito si otterrà il risultato visibile in **Figura 3.1**, cioè durante l'invio del voto effettuato dall'utente con un click sull'apposito link, verrà visualizzato un box che avverte l'utente dell'attività in esecuzione. Anche in questo caso il risultato è sicuramente d'effetto ed il codice richiesto risulta essere veramente minimale.
Oltre a questi controllo e pochi altri inclusi direttamente nel "core", ASP.NET Ajax Extensions mette a

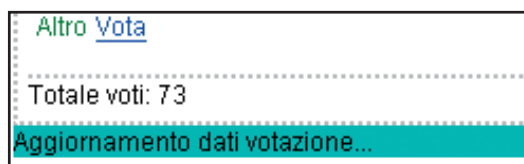



Fig. 3.1: La toolbox dei nuovi controlli Ajax

disposizione una libreria aggiuntiva di controlli server pronti all'uso. Ad esempio in una applicazione CMS, la possibilità di esprimere un giudizio ad un articolo votandolo è ormai all'ordine del giorno; infatti tutti almeno una volta avranno visto a fianco di un articolo in Internet, una serie di stellette più o meno colorate in relazione al giudizio espresso dagli utenti per quell'articolo. Per aggiungere questa funzionalità esiste nell'Ajax Toolkit un apposito controllo che, una volta inserito nella nostra applicazione (in particolare all'interno del controllo BoxArticolo.ascx), permette senza scrivere codice, di avere un sistema visuale per la votazione di un articolo. Il codice da aggiungere è il seguente:

```
<ajaxToolkit:Rating
    ID="MyRating"
    runat="server"
    CurrentRating="2"
    MaxRating="5"
    StarCssClass="ratingStar"
    WaitingStarCssClass="savedRatingStar"
    FilledStarCssClass="filledRatingStar"
    EmptyStarCssClass="emptyRatingStar"
```

```
    OnChanged="MyRating_Changed" >
</ajaxToolkit:Rating>
```

Le proprietà specificate servono a definire graficamente il comportamento del controllo come specificato di seguito:

- CurrentRating: il valore inizialmente visualizzato per il controllo
- MaxRating: valore massimo rappresentabile dal controllo
- StarCssClass: classe CSS da utilizzare per la rappresentazione di una unità della scala di votazione
- WaitingStarCssClass: classe CSS da utilizzare per la rappresentazione di una unità della scala di votazione durante l'istante di elaborazione dopo il clic dell'utente
- FilledStarCssClass: classe CSS da utilizzare per la rappresentazione di una unità della scala di votazione quando si trova nello stato selezionato
- EmptyStarCssClass: classe CSS da utilizzare per la rappresentazione di una unità della scala di votazione quando si trova nello stato vuoto

Inoltre è definito un evento lato server per gestire la votazione. Questo viene eseguito sul server nel momento in cui l'utente effettuerà la votazione, permettendo la gestione del voto inviato sia per la memorizzazione che, per esempio, per visualizzare medie sulle votazioni in tempo reale, come nell'esempio implementato visibile in dettaglio in **Figura 5**. Per ottenere questo comportamento però, abbiamo bisogno di utilizzare un nuovo UpdatePanel per poter gestire sul server il cal-

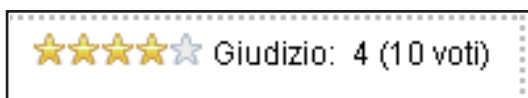



Fig. 5: I dati aggiornati dopo il voto assegnato all'articolo

colo del voto e renderizzare sulla pagina il risultato aggiornato. Per farlo verrà utilizzata una semplice etichetta con nome LblVoti la quale, in seguito ad ogni votazione eseguita con un click del mouse sulle "stelle" del controllo Rating che corrispondono al voto da 1 a 5, dopo un approssimativo calcolo del voto medio e del numero di voti effettuati, verrà aggiornata con questi dati come feedback dell'operazione per l'utente.
Le aggiunte al codice del controllo BoxArticolo. ascx per ottenere il comportamento illustrato sono le seguenti (visibili in Design Mode con Visual Studio 2005 come in **Figura 6**):

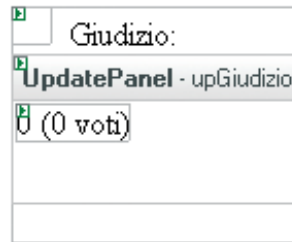



Fig. 6: Visualizzazione di un UpdatePanel a design time in Visual Studio 2005

```
<asp:UpdatePanel ID="upGiudizio" runat="server">
    <ContentTemplate>
        <ajaxToolkit:Rating
            ID="MyRating"
            ...
            AutoPostBack="True" >
        </ajaxToolkit:Rating>
         Giudizio: 
        <asp:Label ID="LblVoti" runat="server"
                                        Text="0
                              (0 voti)"></asp:Label>
    </ContentTemplate>
</asp:UpdatePanel>
```

La modifica principale è l'utilizzo di un altro controllo UpdatePanel chiamato upGiudizio che ora contiene sia il controllo rating descritto in precedenza, sia il controllo Label che sarà utilizzato come esposto in precedenza. Più precisamente il controllo Ajax ha impstato una proprietà aggiuntiva (AutoPostBack="True") che gli consente di effettuare un PostBack come i tradizionali controlli ASP.NET, in modo che nel codice della pagina server l'evento collegato possa gestire i dati inviati dal controllo e visualizzarli nell'etichetta appena inserita.
Il codice dell'evento descritto è abbastanza banale.

```
protected void MyRating_Changed(object sender,
                                    RatingEventArgs e)
{
    Thread.Sleep(400);
    NumVoti++;
    SommaVoti += Convert.ToInt32( e.Value );
    LblVoti.Text = string.Format("{0} ({1} voti)",
                              SommaVoti/NumVoti, NumVoti);
}
```

Il primo comando è semplicemente un ritardo introdotto per simulare una elaborazione dei dati. I due comandi seguenti aggiornano il numero di voti totali e la somma dei valori delle votazioni, in questo caso implementati nella pagina in modo da simulare la memorizzazione dei dati fra un postback e l'altro mentre invece vengono semplicemente mantenuti in una

variabile di sessione; non vuol essere infatti un'implementazione "rigorosa" ma un semplice spunto per fare conoscenza con gli strumenti disponibili. Ne è dimostrazione il fatto che il singolo utente solitamente può votare un articolo una sola volta, mentre in questo esempio questa limitazione non è stata implementata per poter testare il comportamento dei controlli. Da notare comunque nel codice l'utilizzo di e.Value per ottenere il valore del voto specificato dall'utente.
L'ultima riga del metodo descritto visualizza come anticipato le statistiche aggiornate nel controllo Label che, essendo contenuto all'interno di un Updatepanel, verrà aggiornato sul client senza che avvenga il caricamento dell'intera pagina, ottenendo così il risultato desiderato.
Con questo semplice esempio è evidente come sia possibile, con l'utilizzo della libreria Ajax Control Toolkit, dotare le proprie applicazioni di funzionalità Ajax-enabled semplicemente aggiungendo un controllo su di una pagina aspx ed impostando le proprietà necessarie. Inoltre con la versione finale di Ajax Extensions rilasciata recentemente, questa libreria è stata arricchita di ulteriori controlli rispetto a quelli presenti in precedenza che saranno sicuramente apprezzati dagli sviluppatori ASP.NET per la loro utilità. In particolare i nuovi controlli sono:

- Autocomplete, permette di aggiungere ad un semplice TextBox la funzionalità di completamento automatico per suggerire termini durante la digitazione dell'utente (controllo che era originariamente incluso direttamente nel core).
- Calendar, permette di usare un comodissimo calendario popup funzionante lato client (al contrario del controllo nativo ASP.NET che invece funziona per mezzo di postback).
- MaskEdit, permette di inserire testo in modo formattato.
- TabContainer, implementa completamente lato client un controllo a Tab.

DA ATLAS AD ASP.NET AJAX EXTENSIONS

Il cammino di Microsoft verso la versione 1.0 di ASP.NET Ajax Extensions è stato lungo e non privo di intoppi. Primo fra tutti il drastico cambiamento avvenuto fra la versione CTP (Community Tecnology Preview) rilasciata nel luglio 2006 (in cui il nome ufficiale non era ancora stato annunciato ed il prodotto era rilasciato con il nome in codice Atlas) e la prima versione Beta rilasciata agli inizi di Novembre 2006. In questo salto, molte delle funzionalità che erano state sviluppate fino al momento della CTP, sono state tagliate e rimosse dalla versione che sarà poi quella definitiva, creando non pochi problemi a chi aveva già iniziato ad utilizzare la versione precedente per nuovi sviluppi (pensando che le funzionalità sarebbero rimaste pressochè invariate vista l'inclusione da parte di Microsoft nella CTP, di una licenza che permetteva lo sviluppo e la redistribuzione di applicazioni). La versione finale (RTM) su cui si basa il codice di questo articolo, è stata rilasciata il 23 Gennaio 2007.

In conclusione, risulta evidente come con questi nuovi strumenti lo sviluppatore ASP.NET abbia a disposizione nuovi mezzi per rendere le proprie applicazioni più usabili ed interattive, con il grosso vantaggio di poter sfruttare in alcuni casi le potenzialità di Ajax anche senza doversi "sporcare le mani" con la scrittura ed il testing di numerose righe di codice JavaScript. D'altro canto lo sviluppatore più esigente avrà invece a disposizione un framework molto potente in grado di essere sfruttato in modo capillare sia scrivendo codice ad hoc da eseguire sui client (ricordando che in questo caso Ajax Extensions possono essere utilizzate con qualsiasi tecnologia e non necessariamente con il solo ASP.NET), sia sfruttando il modello creato per l'integrazione lato server in ASP.NET della tecnologia Ajax. In particolare per quest'ultimo punto, va segnalato come le classi UpdatePanel, ScriptManager, ecc.. (volutamente utilizzate in modo introduttivo in questo articolo per dare un primo spunto di approfondimento) siano in realtà estremamente complesse e flessibili, tanto da poter tranquillamente richiedere articoli ben più corposi del presente per poter approfondire le potenzialità più avanzate. Il consiglio quindi è quello di dare un'occhiata alla documentazione ufficiale per scoprire come sfruttare in modo avanzato questa nuova piattaforma.
Ma è comunque evidente come, anche senza spingersi troppo nei meccanismi più profondi di questa libreria, Microsoft abbia messo a disposizione uno strumento facile da integrare nelle applicazioni ASP.NET, anche realizzate precedentemente, con sforzo minimo e risultati degni di nota.
Indubbiamente il rilascio di queste estensioni, permetterà un'ulteriore diffusione di questa tecnologia anche tra gli sviluppatori ASP.NET che attualmente sono costretti a rivolgersi a soluzioni di terze parti per aggiungere funzionalità Ajax ai propri siti web. Va inoltre tenuto in considerazione che questo prodotto, almeno da quanto risulta dalla roadmap futura di Microsoft, sarà direttamente integrato in ASP.NET dalle prossime versioni, quindi in futuro tutto il necessario per il suo funzionamento sarà direttamente incluso nell'istallazione standard del framework .Net.

Fabrizio Bernabei

LA COMUNICAZIONE SECONDO .NET 3.0

WCF È LA NUOVA INFRASTRUTTURA SERVICE-ORIENTED PER LO SVILUPPO DI APPLICAZIONI DISTRIBUITE, CHE PERMETTE DI RAGGRUPPARE IN UNA SOLA TUTTE LE VECCHIE TECNOLOGIE PER LA PROGRAMMAZIONE DI RETE. USIAMOLA SUBITO!

La diffusione massiccia delle applicazioni distribuite, dei web services, e la capillarità della rete Internet, hanno senza dubbio sconvolto e modificato il modo in cui si progetta e si sviluppa software al giorno d'oggi. Windows Communication Foundation cerca di creare uno strato comune che risolva molte delle problematiche che si presentano quando si inizia a pensare ad un'applicazione di rete. Far comunicare due o più applicazioni è una questione complessa se si pensa al numero di protocolli, di tecnologie, di approcci, che sono stati sviluppati fino ad oggi, o sarebbe meglio dire fino a prima di WCF.
Soffermandosi sulla piattaforma Microsoft, ci si poteva ritrovare in questi casi di fronte ad una scelta, non sempre esclusiva, fra tecnologie come gli XML Web Services, se si ha bisogno di servizi web multipiattaforma, .NET Remoting per invocazione remota di oggetti .NET, o ancora Enterprise Services, Message Queues, e così via, per non parlare dei diversi protocolli di comunicazione: UDP, TCP, http, ecc.
WCF semplifica lo sviluppo di applicazioni distribuite attraverso un puro approccio service-oriented, e supporta inoltre diversi stili e modelli di programmazione per mezzo della sua caratteristica architettura a strati.

REQUISITI

Conoscenze richieste
conoscenza di base del .NET Framework 3.0

Software
Visual Studio 2005, .NET Runtime 3.0

Impegno

Tempo di realizzazione

ARCHITETTURA DI WCF

La figura 1 rappresenta l'architettura a layer caratteristica di WCF, che permette l'approccio a diverse tipologie di applicazioni connesse in rete.
I Contratti definiscono i vari aspetti di un sistema a messaggi. I messaggi ed i relativi parametri che un servizio può creare o consumare sono descritti dai Data Contracts, definiti per mezzo di schemi XSD. I message contract definiscono le parti di un messaggio utilizzando il protocollo SOAP. Le firme dei metodi di un servizio sono descritti dai service contract.
Policy e bindings rappresentano le condizioni

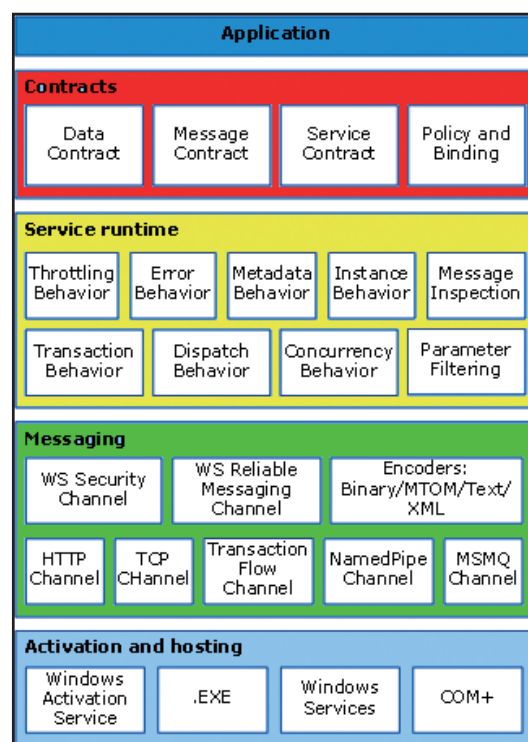



Fig. 1: L'architettura a strati di WCF

necessarie per stabilire una comunicazione con un servizio. Per esempio sarà necessario come minimo specificare il protocollo di trasporto da utilizzare, una codifica dei messaggi, requisiti di sicurezza e così via.
Lo strato sottostante a quello dei contratti è quello di Service Runtime.
Esso, come indica il nome, contiene e specifica il comportamento che caratterizza un servizio durante il suo ciclo di vita, cioè durante la sua esecuzione, e dunque quanti messaggi sono stati processati dal servizio, gli errori verificatisi, le istanze del servizio in esecuzione,
Il terzo strato è quello di Mesaging, composto da canali (channel). Un channel è il componente che si occupa del processamento dei messaggi. Un insieme di canali è detto channel stack.
Mentre lo strato Service Runtime analizza il corpo

di un messaggio, i canali operano sulle intestazioni. Esistono due tipologie di canali, quelli di trasporto e quelli di protocollo.
I primi leggono e scrivono i messaggi da e sulla rete, eventualmente codificando e decodificando.
I canali di protocollo implementano il processamento dei messaggi, per esempio leggendo o scrivendo delle intestazioni addizionali. Lo strato di messaging definisce anche i possibili formati per lo scambio dei dati, per la gestione della sicurezza e dell'affidabilità delle trasmissioni.
Infine, dato che in fin dei conti un servizio è un programma, esso deve essere in qualche modo eseguito. Esistono però diverse possibilità, vale a dire l'esecuzione sotto forma di un'applicazione (self hosted service) oppure ospitare il servizio in un contenitore come IIS o il nuovo WAS (Windows Activation Service) di Vista. Infine un servizio WCF può essere eseguito automaticamente come un qualunque altro servizio di Windows.

WCF IN PRATICA

Agli sviluppatori piace mettere le mani in pasta, e verificare le funzionalità di una tecnologia di sviluppo, e non c'è niente di meglio che inventarsi un'applicazione di esempio, pensarla, progettarla, ed implementarla secondo tale nuova tecnologia. Con WCF si avrebbe l'imbarazzo della scelta riguardo alle possibili applicazioni del mondo reale da implementare. In questo articolo, per semplicità e perché molti conoscono già i requisiti fondamentali si vedrà come WCF permette di creare una Chat multi-utente.
Il primo passo è quello di aprire Visual Studio 2005, creare un nuovo progetto, selezionando come tipologia WCF Service Library.

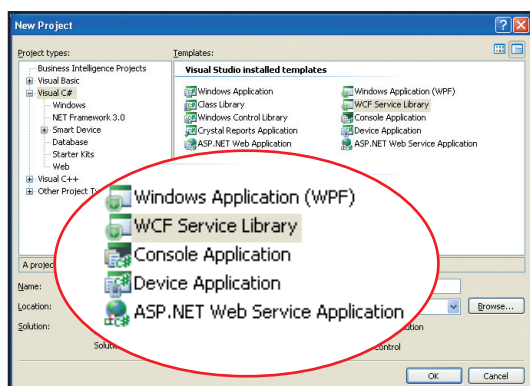



***Fig. 2:* Creare una libreria di classi per WCF**

CREARE IL CONTRATTO

Il contratto della nostra chat definirà le operazioni che il servizio metterà a disposizione dei suoi client. In questo articolo implementeremo per brevità le operazioni fondamentali, quindi quelle che permettono di connettersi alla chat, di spedire e ricevere messaggi, e di disconnettersi al termine dell'utilizzo.
Per definire un contratto bisogna creare un'interfaccia marcata con l'attributo ServiceContract, e i cui metodi, cioè le operazioni del servizio, siano marcati con l'attributo OperationContract.
L'interfaccia si chiamerà quindi IChatService e sarà così dichiarata:

```
[ServiceContract]
public interface IChatService
{
  [OperationContract]
  …
}
```

Il servizio di Chat è un tipico esempio in cui il servizio stesso può svolgere delle attività senza che esse siano iniziate da un client, basti pensare per esempio alla funzione per cui un client può ricevere i messaggi inviati dai client rimanenti. Per aggiungere tale funzionalità al nostro servizio, bisognerà implementare un'interfaccia di callback, ed indicare all'interfaccia IChatService di utilizzarla appunto come contratto di callback, impostando la proprietà CallbackContract
Quindi l'interfaccia IChatService avrà la seguente dichiarazione:

```
[ServiceContract(CallbackContract=typeof(IChatCallback
                                                    ))]
public interface IChatService
{
[OperationContract(IsInitiating=true,IsOneWay=false)]
    string[] Connect(string name);
[OperationContract(IsOneWay=true,IsInitiating=false)]
    void SendMessage(string msg);

    [OperationContract(IsTerminating = true, IsOneWay
                                                = true)]
    void Disconnect();
}
```

mentre l'interfaccia IChatCallback avrà la seguente definizione:

```
public interface IChatCallback
{
      [OperationContract(IsOneWay = true)]
      void ReceiveMessage();

      [OperationContract(IsOneWay = true)]
      void UserEnter(string user);

      [OperationContract(IsOneWay = true)]
```

```
    void UserLeave(string user);
}
```

Noterete che le singole operazioni del servizio e che fanno parte integrante del contratto hanno diversi attributi. Le operazioni con la proprietà *IsInitiating* impostata a true, indicano che tale operazione può iniziare una nuova sessione sul server. IsTerminating con valore true invece indica l'opposto, cioè che l'operazione può terminare la sessione di servizio. Naturalmente ciò vale se le sessioni sono consentite, e quindi in questo caso IChatService ha la proprietà SessionMode impostata al valore Required. Per una spiegazione più dettagliata su una sessione WCF si veda l'apposito riquadro. IsOneWay è una proprietà che permette di indicare se l'operazione è a senso unico, o se può anche restituire un valore. Per esempio l'operazione Connect non è di tipo one-way, perché restituirà, al momento della connessione, gli eventuali altri utenti connessi.

ALCUNE OPERAZIONI

Un data contract è un accordo formale fra il servizio ed i client sui dati che saranno scambiati. In questo modo si potrà comunicare e scambiare messaggi, stabilendo solo il formato di dati da scambiare, e precisamente per ogni parametro o tipo di ritorno si potrà definire quali dati saranno serializzati in XML per essere inviati.
Per esempio possiamo definire il seguente contratto sui dati scambiati per identificare un messaggio, che conterrà il nome dell'utente, il messaggio testuale e l'istante di invio:

```
[DataContract]
public class MsgContract
{
    private string m_User;
    private string m_Message;
    private DateTime m_Time;
    private MessageType m_Type;
    [DataMember]
    public MessageType Type
    {
        get { return m_Type; }
        set { m_Type = value; }
    }
    [DataMember]
    public string User
    {
        get { return m_User; }
        set { m_User = value; }
    }
    [DataMember]
    public string Message
    {
        get { return m_Message; }
        set { m_Message = value; }
    }

    [DataMember]
    public string Time
    {
        get { return m_Time; }
        set { m_Time = value; }
    }
}
```

IMPLEMENTARE IL SERVIZIO

Per sviluppare il servizio basta creare una nuova classe che implementi l'interfaccia IChatService. Dato che il servizio di chat è un servizio multiutente, sarà necessario creare un'istanza dedicata ad ogni utente, utilizzando l'attributo Service Behavior ed in particolare la proprietà Instance ContextMode impostata al valore PerSession, e ConcurrencyMode impostata a Multiple. Da notare, cosa non fatta nell'esempio, che sarebbe necessario gestire eventuali problemi di concorrenza, che possono verificarsi per esempio se due utenti tentassero di connettersi con lo stesso nome.

```
[ServiceBehavior(InstanceContextMode =
                    InstanceContextMode.PerSession,
    ConcurrencyMode = ConcurrencyMode.Multiple)]
public class ChatService : IChatService
{
    public delegate void
            ChatServiceEventHandler(object sender,
                    ChatServiceEventArgs ev);
    public static event ChatServiceEventHandler
                    ChatServiceEvent;
    private string username;
    private static Hashtable users = new
                    Hashtable();
    private ChatServiceEventHandler
                    userEventHandler = null;

    private IChatCallback callback;
    private void CallbackEventHandler(object sender,
                    ChatServiceEventArgs e)
    {
        try
        {
            switch (e.Message.Type)
            {
                case MessageType.MessageReceived:
                    callback.ReceiveMessage(e.Message);
                    break;
                case MessageType.UserEnter:
```

```
                callback.UserEnter(e.Message.User);
                break;
            case MessageType.UserLeave:
                callback.UserLeave(e.Message.User);
                break;
        }
    }
    catch
    {
        Disconnect();
    }
}
```

Il campo *username* rappresenta il nome dell'utente collegato, che sarà inserito nell'Hashtable statica users che contiene l'elenco di tutti gli utenti connessi. Il metodo CallbackEventHandler gestisce e smista tutti gli eventi possibili durante una sessione di chat.

```
#region IChatService Members
public string[] Connect(string name)
{
    if (!users.ContainsKey(name) &&
                    !String.IsNullOrEmpty(name))
    {
        username = name;
        userEventHandler = new
ChatServiceEventHandler(CallbackEventHandler);
        users.Add(name, userEventHandler);
        callback =
OperationContext.Current.GetCallbackChannel<IChatCa
                                    llback>();
        callback.UserEnter(name);

        MsgContract msg = new MsgContract();
        msg.Type = MessageType.UserEnter;
        msg.User = name;
        msg.Time = DateTime.Now;
        ChatServiceEventArgs e = new
                    ChatServiceEventArgs(msg);

        SendMessageToAll(e);
        ChatServiceEvent += userEventHandler;
        string[] listUsers = new
                        string[users.Count];
        users.Keys.CopyTo(listUsers, 0);
        return listUsers;
    }
    else return null;
}
```

Alla connessione di un utente si verifica se il suo username non sia già in uso, quindi viene invocato il metodo *UserEnter* e dopo aver costruito un messaggio con le informazioni sul nuovo utente esso viene spedito in broadcast a tutti i client connessi, per mezzo del seguente metodo:

```
private void
    SendMessageToAll(ChatServiceEventArgs e)
{
    if (ChatServiceEvent != null)
    {

        foreach (ChatServiceEventHandler handler
            in ChatServiceEvent.GetInvocationList())
        {
            handler.BeginInvoke(this, e, new
            AsyncCallback(EndAsyncCallback), null);
        }
    }
}
```

Per spedire un messaggio il servizio utilizza il metodo SendMessage, che crea un nuovo oggetto MsgContract, imposta il suo campo Message e crea un oggetto ChatServiceEventArgs da passare al metodo precedente.

```
public void SendMessage(string message)
{
    MsgContract msg = new MsgContract();
    msg.Type = MessageType.MessageReceived;
    msg.User = this.username;
    msg.Message = message;
    msg.Time = DateTime.Now;
    ChatServiceEventArgs e = new
                    ChatServiceEventArgs(msg);
    SendMessageToAll(e);
}
```

Disconnettersi dal servizio ChatService è altrettanto immediato, basta rimuovere l'utente dalla Hashtable, costruire un MsgContract con Type UserLeave, e spedirlo in broadcast per avvertire tutti:

```
public void Disconnect()
{
    if (this.username != null)
    {
        users.Remove(this.username);
        ChatServiceEvent -= userEventHandler;
        MsgContract msg = new MsgContract();
        msg.Type = MessageType.UserLeave;
        msg.User = this.username;
        msg.Time = DateTime.Now;
        ChatServiceEventArgs e = new
                ChatServiceEventArgs(msg); ;
        this.username = null;
        SendMessageToAll(e);
    } }
```

Date uno sguardo al codice in allegato ed eseguite il servizio in modalità di debug per capire cosa succede dietro le quinte.

ESEGUIRE IL SERVIZIO

Per eseguire il servizio bisogna innanzitutto fornire una configurazione. Prima di tutto scegliamo il protocollo da utilizzare e l'indirizzo al quale il servizio aspetterà le richieste dei client. La configurazione avviene per mezzo del classico file di configurazione App.config:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>

   <behaviors>
    <serviceBehaviors>
      <behavior name="metadataBehavior">
           <serviceMetadata httpGetEnabled="true"
  httpGetUrl="http://localhost:12346/ChatService" />
      </behavior>
    </serviceBehaviors>
  </behaviors>

   <services>
    <service
          behaviorConfiguration="metadataBehavior"
                   name="WCFChat.ChatService">
      <endpoint address="" binding="netTcpBinding"
           bindingConfiguration="chatNetTcpBinding"
       contract="WCFChat.IChatService" />
      <host>
        <baseAddresses>
          <add
 baseAddress="net.tcp://localhost:12345/ChatService"
                                                  />
        </baseAddresses>
      </host>
    </service>
  </services>

  <bindings>
    <netTcpBinding>
      <binding name="chatNetTcpBinding"
                    sendTimeout="00:00:01">
        <reliableSession enabled="true" />
        <security mode="None" />
      </binding>
    </netTcpBinding>

  </bindings>
 </system.serviceModel>
</configuration>
```

All'interno della sezione serviceModel e poi services, viene aggiunto un nuovo elemento service, che identifica il nostro servizio di chat. Nella sottosezione host invece viene definito il baseAddress, cioè l'indirizzo al quale sarà raggiungibile il servizio. Il file di configurazione del servizio può essere scritto a mano, ma è molto più semplice ed immediato utilizzare il Microsoft Service Configuration Editor, presente fra i tool del nuovo SDK.

Nel passo successivo dovremo generare i file proxy per il client ed il relativo file di configurazione, quindi sarà necessario creare un behavior per fare in modo che il servizio metta a disposizione i suoi metadati, pubblicando il WSDL del servizio ad un indirizzo http. La procedura è molto semplice utilizzando il Service Configuration Editor (**figura 2**), all'interno del quale basta creare un nuovo service behavior, ed aggiungere a questo un elemento serviceMetadata. Infine bisogna impostare a true la proprietà HttpGetUrl ed un indirizzo http, per esempio http://localhost/12346/ChatService.

A questo punto, bisogna eseguire il servizio, ed in questo caso lo faremo sotto forma di applicazione Console. Ecco il metodo Main:

```
using (ServiceHost host = new
              ServiceHost(typeof(WCFChat.ChatService)))
{
          try
          {
               host.Open();
               Console.WriteLine("ChatService in
                          esecuzione all'indirizzo {0}",
                       host.BaseAddresses[0].ToString());
               Console.WriteLine("Premi un tasto per
                                fermare il servizio...");

               Console.ReadKey();
               host.Abort();

          }
          catch (Exception ex)
          { Console.WriteLine(ex.Message);  }
          finally
          {
             if (host != null &&
               host.State==CommunicationState.Opened)
                 host.Close();
          }
}
```

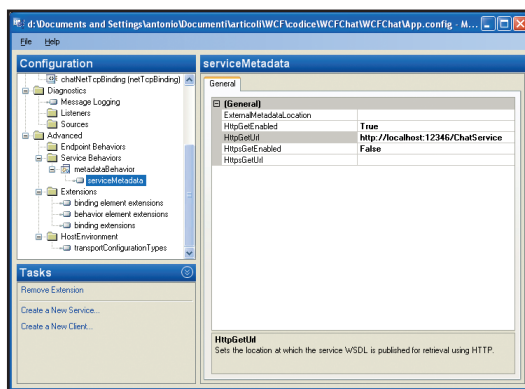



Fig. 3: Il Service Configuration Editor

Si crea un ServiceHost, utilizzando nel costruttore il tipo ChatService, e poi si invoca il metodo Open. Non serve altro, ed il servizio è in ascolto sulla porta 12345.

CREARE UN CLIENT

Se tutto è andato bene fino al passo precedente, il servizio sarà in esecuzione, mentre l'indirizzo definito nel behavior per i metadati ci servirà per configurare e creare il client. In questo caso ci viene in aiuto un altro tool fornito con l'SDK: svcutil.
Con il servizio in esecuzione, aprite un prompt dei comandi di visual studio e digitate il comando seguente dalla directory nel quale si vuol creare il client ed il relativo app.config.

```
svcutil /language:C# /async /config:app.config
                    http://localhost:12346/chatservice
```

Il tool ricaverà il WSDL del servizio, e genererà i file ChatService.cs e app.config, da aggiungere all'applicazione client, che in questo esempio sarà un'applicazione Windows. Lo switch async permette di generare le firme dei metodi sia in versione sincrona che asincrona. In questo modo, quando per esempio un client chiede la connessione al servizio, sarà possibile continuare l'esecuzione del programma senza bloccarsi nell'attendere subito una risposta. Dall'applicazione client è ora semplicissimo utilizzare il servizio, e la maggior parte del tempo sarà necessaria per creare l'interfaccia grafica. Supponiamo di aver realizzato la nostra finestra, con una RichTextBox per contenere le conversazioni, una ListBox che elenca gli utenti connessi, una TextBox per digitare un messaggio ed un pulsante per inviarlo. La classe derivata da Window dovrà anche implementare l'interfaccia IChatCall back, per poter ricevere le comunicazioni di ritorno, e non inziate dal client, provenienti dal servizio. Nell'esempio si è utilizzato WPF per costruire l'interfaccia, ed è possibile studiare il codice in allegato se si vuol conoscere anche questa nuova tecnologia. Non ne parleremo in quanto prescinde dagli scopi dell'articolo.
La prima operazione necessaria è quella di connessione al servizio di chat, che avviene mediante il menù Chat/Connetti. Poiché abbiamo scelto la modalità di implementazione asincrona, dopo aver costruito un oggetto ChatServiceClient, si invoca il metodo BeginConnect:

```
UsernameWindow wnd = new UsernameWindow();
wnd.ShowDialog();
m_Username = wnd.Username;
try
{
InstanceContext context = new InstanceContext(this);
        m_chatClient = new
                    ChatServiceClient(context);
        m_chatClient.BeginConnect(m_Username,
    new AsyncCallback(OnConnectionCompleted), null);
}
catch (TimeoutException timeout)
{
        MessageBox.Show(timeout.Message);
        m_chatClient.Abort();
}
catch (CommunicationException commProblem)
{
MessageBox.Show(commProblem.Message);
        m_chatClient.Abort();
}
```

Se tutto va a buon fine, verrà eseguito il metodo di callback indicato nella chiamata a BeginConnect:

```
private void OnConnectionCompleted(IAsyncResult iar)
{
    try
    {
        string[] users = m_chatClient.EndConnect(iar);
    }
    catch (Exception e)
    {
        MessageBox.Show(e.Message);
    }
}
```

Con l'array degli utenti connessi alla chat si può ora riempire la listbox a destra. Per inviare un messaggio, basta scriverlo nell'apposita TextBox e premere il pulsante Invia:

```
void btInvia_Click(object sender, RoutedEventArgs e)
{
        if(m_chatClient!=null)
m_chatClient.SendMessage(txtMessage.Text);
}
```

CONCLUSIONI

Nell'articolo si è data un'introduzione della tecnologia WCF di Microsoft, che costituisce, all'interno del .NET framework 3.0, l'infrastruttura e le fondamenta per la programmazione distribuita. Tramite Visual Studio 2005 ed i tool del nuovo SDK di Windows implementare un servizio di chat si è rilevato un compito relativamente semplice. In realtà il grande vantaggio consiste nel lavorare con una piattaforma omogenea trasparente ai vari protocolli di rete.

Antonio Pelleriti

IMMOBILI GESTITI CON CLASSE

PROGRAMMARE APPLICAZIONI SEGUENDO IL PATTERN MVC È GARANZIA DI MANUTEBILITÀ. ESISTONO FRAMEWORK SPECIFICI PER SVILUPPARE CON QUESTA MODALITÀ, STRUTS È PROBABILMENTE IL PIÙ NOTO. IMPARIAMO COME USARLO TRAMITE UN ESEMPIO UTILE

In molti conosceranno Struts 2, si tratta di un framework di supporto allo sviluppo di medie e grandi applicazioni web. Il suo principale pregio è quello di essere progettato secondo il ben noto design pattern model view controller (MVC) .
Il modello è costituito dal codice che ha come scopo quello di rappresentare gli oggetti, le relazioni e le procedure del dominio di business, la vista è costituita dal codice che si occupa di mostrare all'utente il modello visualizzando ad esempio un oggetto in dettaglio oppure dei dati aggregati mentre il controller è costituito da quel codice che raccoglie gli input inseriti dall'utente, li valida, invoca le operazioni sul modello, eventualmente modificandolo, e mostra i risultati selezionando la vista opportuna.

ARCHITETTURA LOGICA

Struts propone il concetto di action che rappresenta una qualsiasi operazione eseguita dall'utente. Ad esempio un submit di una form costituisce una action così come il salto ad una pagina per visualizzare il dettaglio di un oggetto. L'action ha prima di tutto la responsabilità di controllare la correttezza degli eventuali parametri utilizzati nell'invocazione.
In caso positivo la action ha l'ulteriore responsabilità di determinare gli oggetti di business che la gestiscono ed incorporare quelli che potranno servire poi alla vista. In maniera dichiarativa, attraverso un file xml, ad ogni action è associata una vista che quando viene infine richiamata può accedere ai dati di business necessari senza doversi preoccupare di come siano stati ottenuti. Vedremo che tutto questo può essere implementato con poche righe di codice

APPLICAZIONE DI ESEMPIO

Nei prossimi articoli riguardanti Struts 2 svilupperemo un'applicazione web dedicata agli intermediari immobiliari attraverso la quale sarà possibile gestire l'insieme delle offerte che un'ipotetica agenzia desidera proporre al

COME INIZIARE

Installare tomcat
Per lo sviluppo dell'applicazione utilizzeremo il web server Tomcat.
Collegatevi all'indirizzo http://tomcat.apache.org/download-55.cgi e salvate il file compresso di Tomcat, in versione zip o tar.gz sul vostro pc.
Scompattate il file in una cartella della quale avrete preso nota del percorso. Ad esempio si supponga di aver scompattato i file nella seguente cartella:
/opt/javaDevEnv/webserver/apache-tomcat-5.5.20

Installare Struts2
Scarichiamo ora la libreria Struts2. Collegatevi all'indirizzo http://people.apache.org/builds/struts/ per accedere all'elenco delle ultime build di Struts 2.
Ogni build ha la propria cartella. Al momento della stesura dell'articolo la versione piu' recente e' la 2.0.1. Entrate quindi nella cartella "2.0.1" elencata nel sito e scaricate il file struts-2.0.1-all.zip che contiene tutto quello di cui avremo bisogno.

Installare Spring
Struts 2 richiede anche Spring. Collegatevi all'indirizzo http://www.springframework.org/ e seguite il link verso la pagina download. Gli esempi sviluppati in questo e nei prossimi articoli sono basati sulla versione 2.0. Al download verrete portati su Sourceforge e avrete la possibilità di scegliere due file da scaricare. Scaricate spring-framework-2.0-with-dependencies.zip che contiene tutti i jar necessari. Questo ci risparmierà ulteriori ricerche.

pubblico. Gli utenti d'altra parte potranno ricercare tra le offerte quelle che più si adattano ai loro desideri.
L'applicazione è volutamente dotata di poche e chiare funzionalità in modo che ci si possa soffermare sugli aspetti tecnologici ed implementativi.

PRIMI PASSI

La prima cosa da fare è creare una struttura fisica sull'hard disk che ci consenta di lavorare secondo i parametri imposti da Struts2. Creiamo prima di tutto una cartella che conterrà tutti i file dell'applicazione, nel nostro caso la chiameremo "demo". La cartella contenente una web application deve essere strutturata secondo le regole definite dagli standard J2EE. La nostra struttura avrà dunque la seguente forma

```
/demo
        /src
        /WEB-INF
                /classes
                /lib
```

La cartella */src* conterrà i sorgenti Java che andranno compilati nella cartella /classes
La cartella */lib* deve contiene i jar necessari all'applicazione. In particolare sarà necessario copiare all'interno di questa cartella i seguenti file estratti dalla distribuzione di Struts 2

- commons-logging-1.0.4.jar
- ognl-2.6.7.jar
- struts2-api-2.0.1.jar
- xwork-2.0-beta-1.jar
- freemarker-2.3.4.jar
- struts2-core-2.0.1.jar

Sempre nella cartella lib posizionate anche il file estratto dalla distribuzione di Spring

- spring.jar

commons-logging è una libreria che offre un'interfaccia omogenea a varie librerie di logging quali log4j o java.util.logging.
OGNL è un linguaggio interpretato che viene utilizzato da alcune tag libraries fornite con Struts 2. XWork è un pattern per lo sviluppo di applicazioni basate su comandi. Struts 2 lo utilizza per la gestione delle action.
Freemarker è un template engine open source. Strut2-core e struts2-api sono le librerie vere e proprie di Struts.
Spring.jar è la libreria di spring. Nella cartella WEB-INF posizionate il file struts-tags.tld. Questo file può esser estratto dal jar di Struts 2 e contiene le informazioni riguardanti le tag library. Se tutto questo lavoro di configurazione vi spaventa, nel CD allegato alla rivista potete trovare la cartella già organizzata con tutto ciò che serve per partire con lo sviluppo. È tuttavia comunque necessario configurare Tomcat perchè carichi la web application all'avvio. Per fare ciò portatevi nella cartella *<tomcat_home>/conf/Catalina/localhost*
e create un nuovo file realestate.xml con il seguente contenuto:

```
<Context docBase="<directory della web
                application>" privileged="false"
        path="/realestate" reloadable="true">
    <Logger
            className="org.apache.catalina.
                    logger.SystemErr
            Logger" verbosity="4"/>
</Context>
```

IL PATTERN MODEL VIEW CONTROLLER

Con il termine "pattern" si indica una soluzione studiata e ben definita per un certo problema. Il pattern Model View Controller: abbreviato MVC può considerarsi come il padre di tutti i pattern. Non che gli altri siano meno importanti, ma elaborare un'applicazone secondo questo pattern la rende automaticamente molto più facilmente gestibile. L'idea è quella di dividere il codice secondo aree di infuenbza ben precise. Prima di tutto il model implementerà le classi e la logica di business dell'applicazione. In questa sezione del codice sarà necessario implementare tutte quelle parti che servono a reperire e gestire i dati. La View si occuperà semplicemente di fornire un output per i risultati provenienti dalla logica di business. Mentre il controller si occuperà di gestire il flusso dell'applicazione. Realizzare un software tenendo bene in mente questa logica così rigida di divisione delle parti è assoluta garanzia di manutebilità. Per progetti di grandi dimensioni ma spesso anche per quelli che hanno anche sole poche righe di codice è indispensabile sviluppare secondo questa modalità. Struts2 implementa totalmente questa logica ed offre al programmatore un ambiente già predisposto e facilitato

IL MODELLO DELL'APPLICAZIONE

Il livello business sarà mantenuto volutamente semplice. La classe principale *Portfolio* da accesso a tutti le offerte immobiliari gestite dall'intermediario.
In dettaglio l'interfaccia esposta dalla classe Portfolio sarà la seguente

```
public final class Portfolio {
        public static Portfolio instance(){...}
```

```
public Set<Proposal> find(Search
                              search){...}
}
```

Il metodo *instance()* permette di ottenere l'unica istanza di Portfolio che manterrà tutte le proposte immobiliari, mentre il metodo *find* accetta un oggetto di tipo *Search* che rispecchia la form di ricerca impostata dall'utente ed estrae un insieme di "*Proposal*" che ne soddisfino i vincoli.
La classe Proposal è un semplice POJO che rappresenta una proposta immobiliare presentata all'utente.

DISEGNARE LA FORM DI RICERCA

Creiamo la pagina Index.jsp che conterrà la form di ricerca degli appartamenti. Per ora permetteremo esclusivamente di ricercare offerte immobiliari in un certo intervallo di prezzo.

```
<%@ taglib prefix="s" uri="/struts-tags" %>
...
<html>
<head>...</head>
<body>
        Cerca tra le offerte
        <s:form action="Search"
                              method="POST">
                <s:actionerror/>
                <s:textfield label="Valore
                                        Minimo"
        name="minValue" tooltip="Valore minimo
                              dell'immobile."/>
                <s:textfield label="Valore
      Massimo" name="maxValue" tooltip="Valore
                                      massimo
                              dell'immobile."/>
                <s:submit/>
```

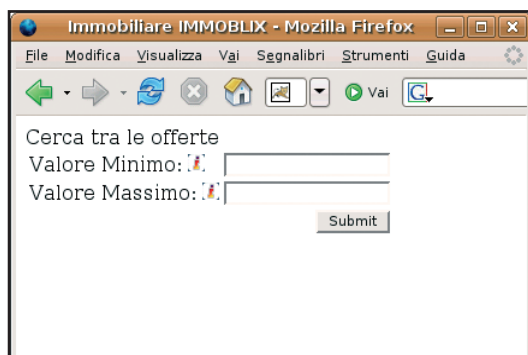



***Fig. 1:** La form che abbiamo realizzato con Struts 2 è automaticamente impaginata e presenta a fianco dei campi una piccola immagine che visualizza il tooltip con un messaggio d'aiuto relativo.*

```
        </s:form>
</body>
</html>
```

La prima riga specifica l'utilizzo delle taglib di Struts definite nel file /struts-tags.tld La form non viene definita attraverso l'usuale tag html <form> ma attraverso l'equialente <s:form>. Il nome dell'azione che Struts2 eseguirà all'invio della form è "Search", come specificato nell'attributo action. In questo caso "Search" non è un URL, ma il nome di un'azione Struts. La form definisce due campi chiamati rispettivamente minValue e maxValue. L'etichetta dell'elemento è specificata all'interno del campo stesso tramite l'attributo label. È anche possibile definire un tooltip che apparirà a fianco del controllo quando il mouse lo sfiorerà. In questi due campi l'utente immetterà rispettivamente gli importi minimi e massimi dei prezzi degli immobili a cui è interessato. Il tag <s:actionError> serve a riportare tutti gli eventuali errori logici riscontrati nella form di ricerca, ad esempio il valore minimo maggiore di quello massimo.

IMPLEMENTARE L'AZIONE

Implementiamo ora l'azione, di cui riportiamo di seguito il codice essenziale:
package com.danidemi.realestate;

```
import ...
public class Search  extends ActionSupport {
        private Integer minValue; private Integer
                                        maxValue;
        private Set<Proposal> foundProposals;

        public Integer getMaxValue() {
                return  maxValue;
        }

        public void setMaxValue(
                        Integer  maxValue)
        {
                this.maxValue = maxValue;
        }

        public Integer getMinValue()
        {
            return minValue;
        }

        public void setMinValue(Integer
                                        minValue)
```

```
                {this.minValue = minValue;}

    public String execute() throws Exception
    {
        if(minValue > maxValue) {
            addActionError("Il
                valore minimo deve essere
                inferiore a quello  massimo.");
            return INPUT;
        }
        Portfolio portfolio = Portfolio.instance();
        foundProposals = portfolio.find(this);
        return SUCCESS;
    }

    public Set<Proposal>
                getFoundProposals(){
        return foundProposals;
    }
}
```

La classe si chiama *Search*, come l'azione invocata dalla form. Questa è una buona regola di naming che consiglio di seguire.
L'azione deve esporre i setter e getter per le proprietà *minValue* e *maxValue* definite nella form. Tramite questa ulteriore regola di naming Struts è in grado di assegnare alle proprietà del bean i valori immessi nei campi.
Il metodo principale è *execute()* che viene invocato da Struts 2 quando l'azione deve essere eseguita. Viene controllato che il valore minimo sia inferiore a quello massimo. Se così non fosse il metodo aggiunge un messaggio di errore alla action attraverso la chiamata al metodo addActionError() e termina restituendo la costante INPUT ad indicare come sia necessario richiedere nuovamente la compilazione della form. Se i parametri immessi sono corretti l'azione interroga l'oggetto del model Portfolio per ottenere l'elenco delle proposte che soddisfano i requisiti e termina con SUCCESS ad indicare coma l'azione sia stata completata con successo.

VALIDAZIONE AUTOMATICA

E' possibile notare come nella action sia eseguito solo un controllo logico sui valori dei campi delle form. Ma come è possibile visto che una web application tipicamente è in grado di estrarre le informazioni contenute in una form solo come stringhe ?. E non dovremmo forse aggiungere dei controlli per verificare che i due importi siano ad esempio

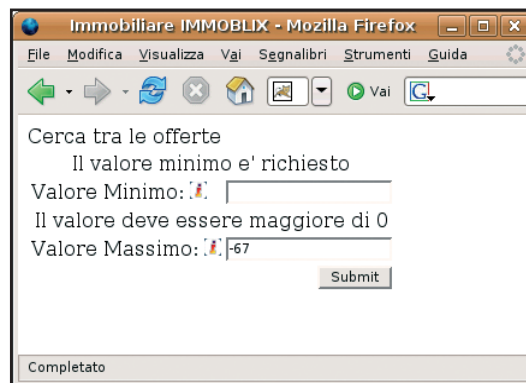



***Fig. 2:** Grazie ai validatori è Struts 2 ad occuparsi del controllo dei valori inseriti e della gestione degli eventuali messaggi di errore.*

maggiori di 0 ? Anche in questo caso Struts 2 ci viene in aiuto. Nella stessa directory nella quale è presente il file Search.java aggiungiamo un nuovo file xml con nome Search-validation.xml che viene utilizzato per specificare il tipo dei parametri della form e i controlli da eseguire.

```
<!DOCTYPE ... >
<validators>
    <field name="maxValue">

        <field-validator type="required">
            <message>Il valore massimo e'
                            richiesto</message>
        </field-validator>

        <field-validator type="int">
            <param name="min">0</param>
            <message>Il valore deve essere
                maggiore di ${min}</message>
        </field-validator>
    </field>

    <field name="minValue">
        [...]
    </field>

</validators>
```

Per ogni campo della form caratterizzato dai tag field è possibile specificare una serie di validatori tramite il tag field-validator. All'interno di questo è poi possibile specificare il messaggio di errore da visualizzare nel caso il campo non passi la validazione. Nel file sono utilizzati i validatori "required" che impongono che il campo della form sia fornito e il validatore "int" che controlla che il parametro della form sia intero ed eventualmente incluso entro due limiti "min" e "max" definiti attraverso i tag "param" con attributo "min" e "max".

LA PAGINA DEI RISULTATI

La pagina dei risultati Hello.jsp elencherà le proposte immobiliari contenute nella action.

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<!DOCTYPE ... >
<html>
        <head>...</head>
    <body>
                <s:if
                 test="foundProposals==null">
                        Nessuna offerte
            trovata rispondenti ai criteri di ricerca
                </s:if>
                <s:else>
                        Offerte trovate
                 rispondenti ai criteri di ricerca
                        <table>
                      <s:iterator
                      value="foundProposals">
                        <tr>
<td><s:property value="description"/></td>
<td><s:property value="value"/></td>
                        </tr>
                      </s:iterator>
                        </table>
                </s:else>
    </body>
</html>
```

Nella pagina dei risultati i tag <s:if> e <s:else> riproducono la funzione di un costrutto if else in Java. In dettaglio la condizione dell'if è costruita utilizzando la notazione OGNL foundProposals==null. FoundProposals viene valutato come il risultato dell'invocazione di *getFoundProposals*() sull'action corrente. Lo scopo è quello di visualizzare un messaggio appropriato nel caso non ci siano proposte immobiliari che soddisfino i requisiti. Nel ramo else invece realizziamo una semplice tabella che indica tutte le proposte trovate. Il costrutto <s:iterator> fa si che il codice contenuto venga rieseguito per ogni oggetto contenuto nella Collection indicata. Tra il tag <s:iterator> e </s:iterator> ci si può riferire alle proprietà dell'oggetto corrente della collezione. In questo caso si scrivono le proprietà description e value.

COMBINARE IL TUTTO

Ora abbiamo la pagina di raccolta dei dati, l'azione da attivare e la vista. Come mettere tutto insieme Struts utilizza per questo compito il file struts.xml.

```
<!DOCTYPE ...>
<struts>
        <include file="struts-default.xml" />
        <package name="default"
                  extends="struts-default"><action
name="Search"class="com.danidemi.realestate.S
                                        earch">
                    <result
             name="success">/Hello.jsp</result>

                    <result
        name="input">/Index.jsp</result>
                  </action>
        </package>
</struts>
```

Questo file specifica le classi delle azioni e la vista a cui deve essere inviata l'azione in concomitanza dei diversi risultati possibili. Il primo elemento importa alcune impostazioni di default che in casi standard è sempre meglio utilizzare. L'elemento package racchiude una serie di actions logicamente collegate. Per ora l'unica action presente e' proprio quella della ricerca, specificata attraverso l'elemento action. Qui sono specificati il nome dell'azione e la classe che Struts2 istanzierà ed eseguirà.
L'elemento action a sua volta contiene una serie di elementi result che specificano le viste a cui girare l'azione una volta completata l'esecuzione ed ottenuto il risultato dell'operazione. La stringa specificata nell'attributo result è proprio la stringa restituita dalla action. I percorsi delle viste sono specificati a partire dalla root della web application. In questo modo è possibile estendere il meccanismo di redirezione semplicemente restituendo un opportuno parametro. Per testare l'applicazione far partire Tomcat e collegarsi all'indirizzo http://127.0.0.1:8080/realestate/Index.jsp

CONCLUSIONI

Apparentemente il meccanismo di gestione di Struts può apparire complesso, tuttavia la pratica insegna che dopo un primo approccio spesso lento, non appena ci si è impratichiti con il meccanismo di funzionamento, si ottiene un notevole innalzamento della velocità di sviluppo. Oltre a questo, come più volte ripetuto, sviluppare con struts è sinonimo di aderenza al pattern MVC, ed in tal senso ogni applicazione ne trae enormi benefici.

Daniele De Michelis

VEDERE FLICKR DAL CELLULARE

IN QUESTO ARTICOLO CREEREMO UN'APPLICAZIONE PER CELLULARI CHE CONSENTE DI RICERCARE E VISUALIZZARE IMMAGINI PRELEVATE DAL FAMOSO "FLICKR". A MARGINE MOSTREREMO COME CREARE UN SERVER PHP CHE INTERAGISCE CON IL CELLULARE

Flickr è un servizio Web che permette di condividere foto tra diversi utenti. Ognuno può creare una propria galleria di immagini e permettere ad altre persone di visualizzare le proprie foto. Molti sviluppatori, basandosi sul servizio e la notorietà di Flickr, hanno realizzato altri servizi. Ad esempio sul sito http://incubator.quasimondo.com/flash/flickeur.php potete trovare FlickEur, un servizio che mostra in streaming diverse foto random, prese appunto da Flickr. Su piattaforma J2ME è possibile utilizzare MobUp (http://www.mobup.org/), un programma che permette di effettuare l'upload di immagini su Flickr direttamente dal cellulare. Quello che realizzeremo in questo articolo è un programma che riesca ad effettuare ricerche su Flickr da piattaforma mobile e che permetta di visualizzare le immagini corrispondenti ai criteri di ricerca sullo schermo del nostro cellulare. Il seguente diagramma mostra l'architettura per la nostra applicazione.

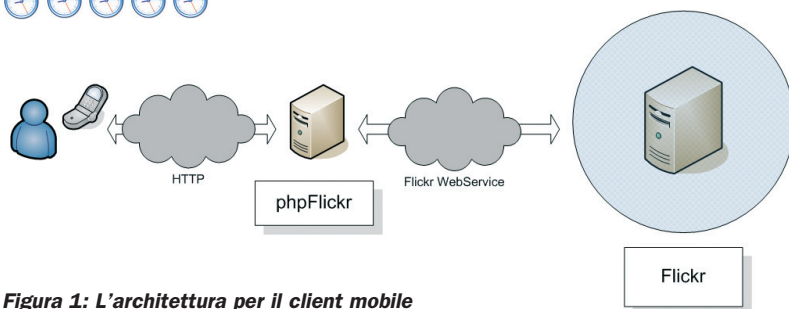



Figura 1: L'architettura per il client mobile

Il primo passo consisterà nella realizzazione di alcune pagine che permettano di effettuare ricerche utilizzando le API esposte da Flickr. Per questo primo passo ci avvarremo di PHP. Per interfacciare PHP con flickr utilizzeremo phpFlickr (http://www.phpflickr.com/), una libreria opensource che ci renderà più agevole il lavoro. Dopo aver effettuato una ricerca dovremo trovare un modo per restituire al cellulare le informazioni corrispondenti alle immagini trovate. Infine dovremo creare sul cellulare, in J2ME, un parser che analizzi le informazioni recuperate e consenta finalmente di visualizzare le immagini.

Dal punto di vista del server dobbiamo sviluppare, sempre in PHP, una pagina che permette di ridimensionare le foto scaricate. Per farlo utilizzeremo le classiche librerie GD, presenti nella distribuzione di PHP. Dal punto di vista del cellulare dobbiamo creare un'applicazione che permette all'utente di scegliere una parola da ricercare. Successivamente questa applicazione deve essere in grado di comunicare con le pagine PHP che effettuano la ricerca e parsare le informazioni che vengono restituite. Infine sarà necessario scaricare le foto, ridimensionate dalla pagina PHP, e visualizzarle a schermo.

LA RICERCA CON PHPFLICKR

PhpFlickr è una classe sviluppata da Dan Coulter in PHP4 che agevola l'accesso alle API offerte da Flickr. Quest'ultimo offre una serie di servizi che possono essere sfruttati utilizzando un API XML molto semplice. Le funzionalità che possono essere usate vengono riportate di seguito:

- Informazioni su commenti relativi alle foto
- Upload di foto
- Gestione delle proprie foto
- Gestione dei favoriti
- Ricerca di persone per foto/foto/email/username
- Gestione della localizzazione delle foto (tag con Geolocalizzazione)

Per utilizzare le API di Flickr dobbiamo ottenere una key, da includere nel nostro codice.

Questa ci permette di effettuare le chiamate verso la piattaforma Flickr ed ottenere una risposta valida. Per ottenere una key è necessario iscriversi alla pagina http://www.flickr.com/services/api/keys/. Una volta ottenuta la chiave possiamo implementare il nostro codice PHP

```
<?
require_once("phpFlickr.php");
$f = new phpFlickr("CHIAVE_DI_FLICKR");

$text=$_GET['testo'];
$x=$_GET['x'];
$y=$_GET['y'];
$x=$x-10;
$y=$y-10;
$per_page=10;
$page=$_GET['page'];

$photos_list = $f->photos_search(array("text"=>$text,"per_page"=>$per_page,"page"=>$page));

$counter=0;

foreach ($photos_list['photo'] as $photo) {
//      echo "<br>Server id: ".$photo['server']." ";
//      echo "<br>Photo id: ".$photo['id']." ";
//      echo "<br>Secret id: ".$photo['secret']."<br>";
//      echo "<br>Testo: ".$photo['title']."<br>";
$url="http://www.javastaff.com/flickr/resize.php?x=".$x."&y=".$y."&src=http://static.flickr.com/".$photo['server']."/".$photo['id']."_".$photo['secret']."_m.jpg";
        echo "<u>".$url."</u><t>".$photo['title']."</t>";
}
?>
```

All'inizio della pagina PHP richiamiamo la classe di phpFlickr per poter utilizzare i suoi metodi all'interno del nostro codice. La prima cosa che facciamo è creare un oggetto phpFlickr, passando come parametro la chiave che ci è stata fornita da Flickr. Il passo successivo è raccogliere i parametri per la ricerca che ci vengono passati tramite semplici valori dalla GET http. Le informazioni che ci interessano per effettuare la ricerca sono il testo da ricercare e la pagina corrente di visualizzazione. Oltre a queste informazioni la midlet J2ME ci fornirà le dimensioni del display del suo dispositivo, per poter, in una fase successiva, ridimensionare le immagini. La ricerca, che viene effettuata utilizzando il semplice metodo photos_search(), restituisce un array associativo dal quale possiamo estrarre tutte le informazioni che ci servono. Indicando come parametro "text" abbiamo deciso di ricercare sia nelle descrizioni delle foto che nei tag inseriti per catalogarle. Altri due parametri che utilizziamo nelle nostre ricerche sono "page" e "per_page", che ci permette rispettivamente di decidere quale pagina dei risultati vogliamo visualizzare e quanti elementi per pagina. Allo stesso modo potremmo anche ricercare su Flickr delle immagini con riferimenti geografici. Infatti se scorriamo la lunga lista di API di Flickr vediamo che esiste la possibilità di effettuare la ricerca inserendo come parametro un'informazione di localizzazione. Tornando alla nostra applicazione, quando abbiamo recuperato questo array con le informazioni di cui abbiamo bisogno, possiamo cominciare a costruire la risposta per il client. Qui di seguito viene riportato lo schema del formato con cui passeremo le informazioni al client

```
<u>URL</u>
<t>DESCRIZIONE</t>
<u>URL</u>
<t>DESCRIZIONE</t>
```

Praticamente passiamo al client informazioni che riguardano l'url della foto da scaricare e il testo associato alla foto. Ad esempio ricercando la parola "ciao" ci viene restituito dalla pagina PHP le seguenti informazioni

```
...
<u>http://www.javastaff.com/flickr/resize.php?maxx=190&maxy=90&src=http://static.flickr.com/147/361709779_7880901560_m.jpg</u>
<t>Ciao Bella</t>

<u>http://www.javastaff.com/flickr/resize.php?maxx=190&maxy=90&src=http://static.flickr.com/159/361650147_483386b15b_m.jpg</u>
<t>Lombroso @ Le Scimmie - Milano 16.Gennaio.2007</t>
<u>http://www.javastaff.com/flickr/resize.php?maxx=190&maxy=90&src=http://static.flickr.com/146/361626596_84428ce8b8_m.jpg</u>
<t>Toni Luisk Me 16-09-06</t>
...
```

Come potete vedere l'url che viene restituito è quello dell'immagine, con prefisso quello di una pagina PHP. In questo modo effettuiamo il resize dell'immagine.

RESIZE DELL'IMMAGINE

Il ridimensionamento dell'immagine avviene sempre tramite una pagina PHP. Per fare ciò utilizziamo una libreria utilizzabile in PHP, GD.

Questa è una libreria opensource sviluppata in C per la manipolazione delle immagini. Esistono wrapper per diversi linguaggi (Perl, PHP, Python, Ruby etc. etc.), che permettono di utilizzare in maniera molto semplice le funzionalità di GD. Per maggiori informazioni su questa libreria vi rimando al sito ufficiale http://www.libgd.org/. Per vedere se questa libreria è inclusa nella nostra distribuzione PHP dobbiamo semplicemente richiamare la famosa pagina php_info.php. In questa pagina, se il nostro PHP è configurato correttamente, dovrebbe apparire la tabella riportata nella figura.

GD Support	enabled
GD Version	bundled (2.0.28 compatible)
FreeType Support	enabled
FreeType Linkage	with freetype
GIF Read Support	enabled
GIF Create Support	enabled
JPG Support	enabled
PNG Support	enabled
WBMP Support	enabled
XBM Support	enabled

Figura 2: Tabella GD in php_info().php

Se così non fosse dobbiamo configurare la nostra distribuzione con il seguente comando

```
/home/doc/php# ./configure –with-gd
```

Potrebbero mancare delle librerie per poter configurare PHP con GD, quindi nel caso di errori dobbiamo installarle nel nostro sistema e poi ripassare alla configurazione di GD. Ora che abbiamo il supporto GD nelle nostre pagine PHP passiamo al semplice script che ci permette di effettuare il ridimensionamento delle immagini

```
<?php

//Prendiamo i parametri che ci vengono forniti
$x = $_GET["x"];
$y = $_GET["y"];
$image_URL = $_GET["src"];

//Carichiamo l'immagine grazie al suo URL
$image = imagecreatefromjpeg($image_URL);

//Valorizziamo le variabili relative alle dimensioni
                                          dell'immagine
list($width, $height) =
                    getimagesize($source_image_URL);

//Calcoliamo quail devono essere le nuove dimensioni
$resizeW = $width / $x;
$resizeH = $height / $y;
$percent = max($resizeW,$resizeH);
$new_height = round($height /$percent);
$new_width = round($width /$percent);

//Creiamo la nuova immagine
$dest_image = imagecreatetruecolor($new_width,
                                          $new_eight);
imagecopyresampled ($dest_image, $source_image,
                                          0, 0, 0, 0,
 $new_width, $new_eight, $width, $height);

//Trasferiamo l'immagine inserendola come risposta
header("Content-type: image/jpeg");
imagejpeg($dest_image);
imagedestroy($dest_image);
imagedestroy($source_image);
?>
```

Il codice che effettua il ridimensionamento è abbastanza semplice. Una volta che sono stati presi i parametri dalla GET, non facciamo altro che caricare l'immagine, calcolare le nuove dimensioni e restituire la nuova immagine. A questo punto possiamo considerare chiusa la parte server.

RICHIESTA DEL CLIENT

La prima cosa che dobbiamo realizzare dal punto di vista del client è la richiesta iniziale, dove effettuiamo una connessione verso la pagina search.php, alla quale passiamo i dati necessari per la ricerca. Per avere l'informazione sul testo da cercare dobbiamo ricevere l'input digitato dall'utente, utilizzando un classico Form

```
…
f = new Form("Flickr Mobile Search");
testo=new TextField("Text to search",null,30,0);
f.append(testo);

f.addCommand(searchCommand);
f.addCommand(exitCommand);
f.setCommandListener(this);
display.setCurrent(f);
…
```

Nel momento in cui l'utente clicca il tasto per inviare la richiesta dobbiamo prendere il testo inserito e far partire un Thread che effettui la connessione. Questo è possibile implementanto il metodo commandAction() dell'interfaccia CommandListener

```
public void commandAction(Command c,
                                        Displayable s) {
    if (c == exitCommand) {
        destroyApp(false);
        notifyDestroyed();
    }
    else if (c == searchCommand) {
        f = new Form("Flickr Mobile Search");
        f.append("Searching...");
        searched=testo.getString();
        display = Display.getDisplay(this);
        display.setCurrent(f);
        new
      FlickrThread(this,testo.getString(),page).start();
    }

    else if (c == nextPageCommand) {
        f = new Form("Flickr Mobile Search");
        f.append("Searching...");
        display = Display.getDisplay(this);
        display.setCurrent(f);
        page++;
        new FlickrThread(this,searched,page).start();
    }

    else if (c == exit) {
        exit();
    }

    else
        createHome();
}
```

Il fatto di dover far partire un Thread per gestire la connessione HTTP è necessario perché non possiamo inserire all’interno dei metodi richiamati dall’interfaccia grafica (come command Action()), del codice che può essere bloccante, come appunto è quello relativo alla connessione verso il server. Il Thread non deve far altro che aprire la connessione HttpConnection verso la pagina search.php e comunicare le informazioni da ricercare

```
import java.io.*;
import javax.microedition.io.*;
import java.util.*;

public class FlickrThread extends Thread {

    FlickrMobile fm;
    private String phpUrl =
            "http://www.javastaff.com/flickr/search.php";
    private String testo;
    private int page;

    public FlickrThread(FlickrMobile fm,String testo,int
                                                page) {
```

```
        this.fm=fm;
        this.testo=testo;
        this.page=page;
    }

    public void run() {
        try {
            DummyCanvas test=new DummyCanvas();
            int width=test.getWidth();
            int heigth=test.getHeight();
            HttpConnection hc =
(HttpConnection)Connector.open(phpUrl+"?page="+p
age+"&x="+width+"&y="+heigth+"&testo="+testo);
            InputStream is = hc.openInputStream();
            int ch;
            long len = hc.getLength();
            String temp="";
            if(len!=-1) {
                for(int i = 0;i<len;i++)
                    if((ch = is.read())!= -1)
                        temp+=((char) ch);
            }
            else {
                while ((ch = is.read()) != -1)
                    temp+=((char) ch);
            }

            is.close();
            hc.close();
            fm.notify(temp);

        } catch (Exception ex) {
          fm.notify(ex.toString());
        }
    }
}
```

Per avere la risposta dal server leggiamo tutti i caratteri che vengono restituiti dall’InputStream associato alla HttpConnection che abbiamo aperto. Da notare come i dati relativi alla grandezza dello schermo vengano estrapolati dalla classe DummyCanvas, una canvas vuota che viene istanziata soltanto per avere queste informazioni, visto che la classe che estende, Canvas, ha i metodi per sapere altezza e larghezza dello schermo.

PARSING E DOWNLOAD

Una volta che abbiamo finito di scaricare la risposta, la passiamo sotto forma di stringa al metodo notify() della nostra MIDlet

```
public void notify(String result) {
        PacketParser pp=new PacketParser(result);
```

```
        vector=pp.parse();
        if (vector==null) {
            f.append("Error occurred");
            display = Display.getDisplay(this);
            display.setCurrent(f);
        }
        else
        {
            ImageDownloader id=new
                        ImageDownloader(this,vector);
            id.start();
            f = new Form("Flickr Mobile Search");
            f.append("Downloading images...");
            display = Display.getDisplay(this);
            display.setCurrent(f);
        }
}
```

Come abbiamo detto infatti l'applicazione server fornisce come risultato della ricerca uno pseudoXML, che il client deve parsare. Il metodo notify() viene appunto richiamato quando abbiamo ricevuto la risposta per effettuare il parsing delle informazioni. Questo avviene tramite una semplice classe, PacketParser, che utilizzando i classici metodi indexOf() e substring() ricostruisce l'informazione che abbiamo ricevuto come risposta.

```
import java.util.*;

//Formato del pacchetto <u>URL</u><t>TEXT</t>
//URL: Url dell'immagine da scaricare
//TEXT: Testo associato all'immagine

public class PacketParser {

    private String toParse;

    public PacketParser(String toParse) {
        this.toParse=toParse;
    }

    public Vector parse() {
        int start,end;
        String temp="";
        Vector toReturn=new Vector();

        try {
            start=toParse.indexOf("<u>");
            end=toParse.indexOf("</u>");
            while(start!=-1) {
                temp=toParse.substring(start+3,end);
                toReturn.addElement(temp);
                toParse=toParse.substring(end);
                start=toParse.indexOf("<t>");
                end=toParse.indexOf("</t>");
                temp=toParse.substring(start+3,end);
                toReturn.addElement(temp);
                toParse=toParse.substring(end);
                start=toParse.indexOf("<u>");
                end=toParse.indexOf("</u>");
            }
        }
        catch(Exception e) {
        }
        return toReturn;
    }

}
```

Nel caso in cui il vettore restituito da questa classe è null, il metodo notify() che abbiamo visto provvede a segnalare all'utente che si è verificato un errore. Se invece il parsing è stato eseguito correttamente possiamo incominciare a scaricare tutte le immagini. Per fare ciò utilizziamo un'altra classe utile, ImageDownloader, alla quale vengono forniti tutti gli URL delle immagini. Anche in questo caso ricorriamo ad un Thread che effettua la connessione verso la pagina resize.php.

```
import java.io.*;
import java.util.*;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;

public class ImageDownloader extends Thread{

    Vector data;
    Image[] images;
    FlickrMobile fm;

    public ImageDownloader(FlickrMobile fm,
                                    Vector data) {
        this.data=data;
        images=new Image[data.size()/2];
        this.fm=fm;
    }

    public void run() {
        try {
            HttpConnection hc;
            InputStream is;

            for(int i=0,j=0;i<data.size();i=i+2,j++) {
                hc =
               (HttpConnection)Connector.open((String)data.
                                        elementAt(i));
                is = hc.openInputStream();
                images[j]=Image.createImage(is);
                is.close();
                hc.close();
            }
        }
```

```
        catch(Exception e) {
        }

        fm.showImages(images);

    }
}
```

Questo Thread si occupa di scaricare tutte le immagini che sono state trovate dalla ricerca. Per fare ciò viene utilizzato il metodo statico Image.createImage(), che accetta come parametro l’InputStream relativo all’immagine. In questo modo noi non dobbiamo far altro che creare l’HttpConnection con l’URL dell’immagine, ottenere l’InputStream e creare l’immagine. Alla fine dei download, l’array di immagini viene passato alla nostra MIDlet, che visualizza le immagini utilizzando una Form e inserendo dopo ogni immagine la relativa descrizione.

Figura 3: Immagini restituite dal client cercando la parola “vienna”

```
public void showImages(Image[] images) {
        f=new Form("Downloaded images");
        for(int i=0;i<images.length;i++) {
            f.append(new
 ImageItem((String)vector.elementAt(i*2+1),images[i
                                            ],0,null));
        }
        f.addCommand(homeCommand);
        f.addCommand(nextPageCommand);
        f.addCommand(exitCommand);
        f.setCommandListener(this);
        display = Display.getDisplay(this);
        display.setCurrent(f);
}
```

Il risultato della MIDlet può essere visualizzato nell’immagine dell’emulatore.
A questo punto possiamo effettuare un’altra ricerca, facendo visualizzare nuovamente l’interfaccia per l’inserimento dei dati oppure passare alla pagina successiva dei risultati. Per quest’ultima cosa possiamo semplicemente riutilizzare FlickrThread, passando come parametro la pagina incrementata di una unità.

```
...
else if (c == nextPageCommand) {
        f = new Form("Flickr Mobile Search");
        f.append("Searching...");
        display = Display.getDisplay(this);
        display.setCurrent(f);
        page++;
        new FlickrThread(this,searched,page).start();
}
...
```

CONCLUSIONI

In questo articolo abbiamo visto come l’unione di diverse tecnologie può portare a programmi molto interessanti. Utilizzando le API di Flickr siamo riusciti a creare un client mobile, che può fare delle ricerche su Flickr e visualizzare immagini ridimensionate per lo schermo del cellulare. Con lo stesso paradigma, ovvero client J2ME semplice e veloce unito ad una tecnologia agile come PHP, si possono sviluppare molti servizi. Come sempre l’ingegno è la base per i programmi più interessanti. Infine dobbiamo annotare come alcuni dei servizi più interessanti del web mettano a disposizione di noi programmatori una serie di API che possono favorire lo sviluppo di applicazioni client. In molti casi non è necessario reinventare la ruota!

Federico Paparoni

FEDERICO PAPARONI
può essere contattato per suggerimenti o delucidazioni all’indirizzo email federico.paparoni@javastaff.com

IL CODICE LO SCRIVE VISUAL STUDIO

L'IDEA È SEMPLICE: PROGRAMMIAMO UN NOSTRO WIZARD PER LA GENERAZIONE AUTOMATICA DI UN'INTERFACCIA VERSO I DATABASE. VEDREMO CHE L'IMPLEMENTAZIONE OFFRE SPUNTI DAVVERO INTERESSANTI

Nella prima parte di questo articolo abbiamo esplorato la tecnica dei Custom Tools, con i quali possiamo generare automaticamente del codice in Visual Studio.
I generatori di codice sono già presenti in larga misura nell'IDE di Microsoft, basta pensare ai file di risorse, ai dataset, alle Windows Form ecc… tutti casi insomma in cui il programmatore interviene a modificare visualmente qualcosa, ad esempio una Form, e si trova del codice già creato da Visual Studio.
Abbiamo anche esaminato criticamente la generazione del codice associata ai Dataset, e ciò ci ha spinto a creare un nostro strumento per il mapping trasparente di Database, ovvero del codice generato automaticamente, che ci risparmi il lavoro di scrivere le classi che riproducono le tabelle del database e le altre operazioni comunemente collegate.
Ci siamo quindi riproposti di costruire uno strumento di DBMapping, alternativo al meccanismo dei Dataset, forse non migliore ma dove sicuramente possiamo intervenire direttamente nel generare il codice che più ci torna utile.
L'applicazione, che troverete nei sorgenti, è abbastanza complessa e, soprattutto, articolata per cui qui ci limiteremo ad esporre i punti più significativi rimandando ad un'analisi del codice di esempio una lettura più analitica dei vari passaggi; il progetto è sviluppato in Visual Basic, ma i concetti sono naturalmente applicabili anche a C#.

REQUISITI

Conoscenze richieste
conoscenza media di.NET

Software
Visual Studio 2005

Impegno

Tempo di realizzazione

GLI OBIETTIVI

Quello che vogliamo è far sì che, in Visual Studio scrivendo un file XML con qualche semplice informazione su un database si generino automaticamente:

- Un altro file XML con il dettaglio di metadati (informazioni sulla struttura di tabelle, campi, relazioni ecc…) che ci può servire sia come documentazione che come base per applicare trasformazioni con XSL.
- Un file di codice contenente le classi che mappano le tabelle del database con i campi che diventano altrettante proprietà e i metodi che interagiscono in modo trasparente con i dati.
- Altri file, impostabili liberamente dal programmatore che ottenuti attraverso le trasformazioni XSL sul file XML dei metadati.

L'obiettivo primario è comunque quello di ottenere uno strumento leggero, personalizzabile e con di facile apprendimento.

OTTENERE I METADATI

Il primo problema da affrontare è : come ottenere i metadati da un database?
A volersi limitare ad un database specifico le cose sarebbero decisamente più semplici, quasi tutti i database hanno infatti strumenti specifici per effettuare query sui metadati.
Noi però vogliamo che il nostro Tool funzioni con tutte le connessioni standard ADO.NET, quindi con OleDb, ODBC e SqlClient, così da poter mappare database SQL Server, Access ecc…
Dovremmo quindi cercare strumenti che ci consentano di ottenere informazioni sulla struttura di tabelle, viste e, nel caso di SQL Server, di stored procedures. Le classi fondamentali che consentono di interagire con i database seguono uno schema "a strati" abbiamo cioè:

1. Un'interfaccia che definisce le caratteristiche generali; ad esempio per la connessione c'è **System.Data.IDBConnection**, per i comandi **System.Data.IDBCommand** e così via.

2. Una classe di base, nel Namespace **System.Data.Common**, che definiremo, impropriamente, astratta che implementa l'interfaccia; e così avremo **System.Data.Common.DBConnection**, **System.Data.Common.DBCommand** ecc…

3. Le classi che implementano i vari provider; e

quindi *System.Data.SqlClient.SqlConnection*, *System.Data.Oledb.OledbConnection*, *System.Data.Odbc.OdbcConnection*, che ereditano da un'unica classe base.

Nel Namespace *System.Data.Common*, però, vi è anche una classe *DbProviderFactory* che offre la possibilità di attivare gli altri oggetti senza vincolarsi ad uno specifico provider.
La classe ha alcuni metodi molto utili :

Nome	Descrizione
CreateCommand	**Restituisce una nuova istanza della classe del provider che implementa la classe DbCommand.**
CreateConnection	**Restituisce una nuova istanza della classe del provider che implementa la classe DbConnection.**
CreateDataAdapter	**Restituisce una nuova istanza della classe del provider che implementa la classe DbDataAdapter.**
CreateParameter	**Restituisce una nuova istanza della classe del provider che implementa la classe DbParameter.**

DbProviderFactory, a sua volta si costruisce con il metodo condiviso *GetFactory* della classe *Db ProviderFactories*, anch'essa presente in *System.* Data.Common. La funzione GetFactory viene richiamata passando come argomento il nome del provider specifico (che corrisponde a quello del Namespace, ad esempio *System.Data.SqlClient*).
Per richiamare una connessione a SQL Server possiamo quindi scrivere:

```
Dim conn As New SqlConnection(connString)
```

utilizzando il provider specifico, oppure:

```
Dim factory As DbProviderFactory
factory =
 DbProviderFactories.GetFactory("System.Data.SqlClient")
Dim conn As DbConnection
conn = factory.CreateConnection()
conn.ConnectionString = connString
```

utilizzando *DbProviderFactory*.
Con questo secondo metodo avremo come risultato sempre un oggetto *Connection*, ma svincolato dall'utilizzo di un provider di dati specifico, basterà infatti cambiare il nome del provider in *DbProvider Factories.GetFactory* che il rimanente codice rimarrà perfettamente valido.
Fin qui per quanto riguarda il collegamento al database effettuato in modo generico, si tratta adesso di vedere come ottenere i nostri metadati.
Il trucco sta nella funzione *GetSchemaTable* presente in qualsiasi oggetto che implementa l'interfaccia *IDataReader* ovvero il lettore di dati ottenuti attraverso un *Command*. Questa funzione restituisce una *DataTable* con i metadati. Ad esempio:

```
Dim cmd As DbCommand = Factory.CreateCommand
Dim conn As DbConnection
conn = factory.CreateConnection()
conn.ConnectionString = connString
cmd.Connection = conn
cmd.CommandText ="SELECT * FROM [MiaTabella]"
cmd.Connection.Open()
Dim reader As DbDataReader
reader =
 cmd.ExecuteReader(CommandBehavior.CloseConnection)
Dim metadati As DataTable = reader.GetSchemaTable()
reader.Close()
cmd.Connection.Close()
```

Tra i metadati, relativi alle colonne di tabella o di vista, ottenuti con questo metodo ve ne sono alcuni che variano a seconda del database sottostante, ma altri sono presenti in tutti i database, ovvero:

- ColumnName = nome della colonna
- ColumnOrdinal = ordine della colonna
- ColumnSize = dimensione
- NumericPrecision = precisione
- NumericScale = scala
- DataType = tipo di dati .NET espresso dalla colonna
- ProviderType = codice numerico del tipo di dati
- AllowDBNull = consente valori Null
- IsReadOnly = sola lettura
- IsUnique = campo unico
- IsKey = campo chiave
- IsAutoIncrement = campo incrementale

A questo punto non resta che costruire delle classi rappresentino i metadati ottenuti e ci consentano di ottenere facilmente gli schemi di tabelle o viste e relative colonne, lo facciamo nelle classi *Schema Table* e *SchemaColumn* delle quali riportiamo, in **figura 1**, la struttura.
Alla fine avremo :

- una classe *DBSchemaManager* – che ci consente di ricavare degli oggetti *SchemaTable* e *Schema Column* da un database data una stringa di connessione e il nome del provider di dati.
- una classe *SchemaTable* – che contiene il nome della tabella e un insieme di oggetti *Schema Column*
- una classe *SchemaColumn* – che contiene tutte le proprietà comuni relative alla colonna

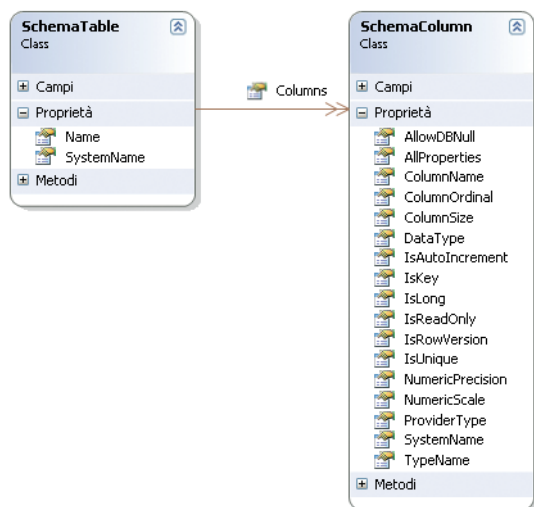



Figura 1: Classi che rappresentano lo Schema di tabelle e colonne

Ovviamente non siamo entrati troppo nel dettaglio, comunque c'è da segnalare che *DBSchemaManager* fornisce anche un metodo per recuperare gli schemi delle stored procedures.

I DATI DI INPUT

Quando, in un progetto, inseriamo un dataset, un file di configurazione ecc… avviene questo processo:

- Visual Studio passa i dati del dataset o del file di configurazione al generatore di codice;
- Il generatore di codice interpreta (parsing) questi dati e genera il codice corrispondente.

A monte abbiamo quindi un file "descrittivo" che serve a generare il codice.
Questo file può presentarsi sia in forma visuale che in forma di codice, per rappresentarlo in forma visuale dovremmo costruire un designer apposito, cosa che esula dal nostro scopo, ci accontenteremo quindi di basarsi su un file XML che editeremo a mano. Questo file XML conterrà la rappresentazione delle informazioni che l'utente fornisce sul database. Mi spiego meglio: ogni volta che dovremo inserire nel progetto il codice di mapping di un database basterà inserire un nuovo file XML sullo stile del web.config e associarlo al nostro Custom Tool che provvederà automaticamente a generare il codice necessario. Poiché uno dei requisiti che ci eravamo proposti è quello di automatizzare il più possibile tutto il processo, dobbiamo far sì che sia possibile ottenere il risultato con il minimo sforzo. Ad esempio, il generatore deve riuscire a lavorare correttamente già partendo da un file di questo tipo:

```
<database name="Northwind" namespace="Northwind"
          type="SqlClient"  connectionString="Data
   Source=Localhost;Initial Catalog=Northwind;Integrated
                                        Security=True">
   <table name="Orders"/>
</database>
```

Cioè, non dovremo mappare obbligatoriamente tutte le colonne della tabella, ma indicare semplicemente il nome. Saranno le classi che recuperano lo schema dal database a completare le informazioni mancanti.
La semplicità però non deve andare a discapito della flessibilità, per cui occorre prevedere anche una modalità più raffinata di mapping dove l'utente può definire con più precisione i dettagli quali : il nome che assumerà la classe corrispondente a una tabella o la proprietà corrispondente a una colonna, come ad esempio:

```
<database name="Northwind" namespace="Northwind"
      type="SqlClient"  connectionString="Data Source=
          Localhost;Initial Catalog=Northwind;Integrated
                                        Security=True">
   <table name="Orders" sysname="Ordine">
      <field keyField="true" name="OrderID"
                                   sysname="Codice"/>
      <field name="Freight" remove="true"/>
   </table>
</database>
```

dove si definisce che la classe corrispondente alla tabella "Orders" si chiama Ordine, il campo "OrderID" è un campo chiave (da utilizzare quindi nelle operazioni di Insert, Update e Delete) e la proprietà si chiamerà Codice, il campo "Freight" non va inserito come proprietà della classe.
Si dovrà anche poter scegliere di effettuare un mapping manuale, prendendo in considerazione solo le colonne esplicitamente indicate:

```
  <table name="Products" sysname="Prodotto"
                              mappingType="explicit">
   <field name="ProductID" keyField="true"/>
   <field name="ProductName"/>
  </table>
```

aggiungendo l'attributo mappingType su "explicit" le proprietà create nella classe Prodotto saranno solo ProductID e ProductName. E così via…
Questo linguaggio di definizione, dovrà avere corrispondenza in altrettante classi del nostro Custom Tool, che vediamo sinteticamente schematizzate nel diagramma in **figura 2**.
A questo punto creiamo la classe InputParser che compie le seguenti operazioni:

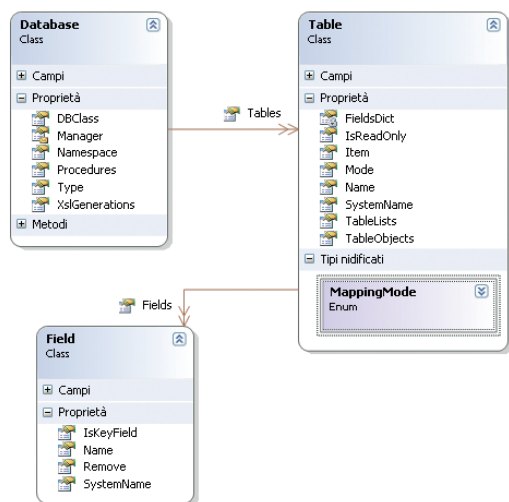



Figura 2: ***Classi che rappresentano il mapping effettuato dall'utente***

1. Deserializza il file XML editato dall'utente nelle classi relative che abbiamo visto nel diagramma in figura 2.
2. Si connette al database utilizzando la proprietà ConnectionString impostata dall'utente nell'elemento database del file XML recuperando le informazioni sullo schema delle tabelle o viste indicate dall'utente nel file di mapping.
3. Effettua un merge tra le informazioni fornite dall'utente e quelle provenienti dalla lettura dei metadati.

UN AIUTO ALLA DIGITAZIONE DEL FILE XML

Vogliamo fornire al nostro utente anche un aiuto nella digitazione del file XML con l'intellisense attivo mentre digita elementi e attributi (vedi **figura 3**), un po' come avviene con i file di configurazione web.config.

```
<database name="Northwind" namespace="Northwi
connectionString="Data Source=SMELZO;Initial
="True">
  <|
    !--          "Products" sysname="Prodotto" n
    ?            e="ProductID" keyField="true"/>
    class        e="ProductName"/>
    procedure
    table        e="Orders" sysname="Order">
    xslGenerator eyField="true" name="OrderID" s
    <list ref="Order Details" sysname="Detail
="true">
      <relation childKey="OrderID" parentKey=
    </list>
    <object ref="Customers" sysname="Customer
"true">
      <relation childKey="CustomerID"
```

Figura 3: ***Intellisense sul file XML***

Per far questo dovremo preparare un file XML Schema (XSD) che descrive la struttura del documento XML di input e che riportiamo in sintesi (l'originale è molto più articolato):

```
<xs:schema attributeFormDefault="unqualified"
                elementFormDefault="qualified"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="database">
    <xs:complexType>
      <xs:sequence minOccurs="1">
        <xs:element maxOccurs="unbounded"
                 minOccurs="0"  name="table">
          <xs:complexType>
            <xs:sequence  maxOccurs="unbounded"
                                  minOccurs="0">
              <xs:element maxOccurs="unbounded"
                   minOccurs="0" name="field">
                <xs:complexType>

                  <xs:attribute name="name"
                    type="xs:string" use="required" />
                  <xs:attribute name="sysname" type="
                         xs:string" use="optional" />
                  <xs:attribute name="keyField"
                    type="xs:boolean" use="optional" />
                  <xs:attribute name="remove"

type="xs:boolean" use="optional" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="name" type="xs:string"
                               use="required" />
            <xs:attribute name="sysname" type="
                         xs:string" use="optional" />
            <xs:attribute name="readOnly"
                    type="xs:boolean" use="optional" />
            <xs:attribute name="mappingType"
                  type="mappingMode" use="optional"/>
          </xs:complexType>
        </xs:element>
</xs:schema>
```

In questo modo basta che l'utente associ lo schema al file che sta editando (nella proprietà Schema del file XML, vedi **figura 4**) che avrà a disposizione l'intellisense in tempo reale.

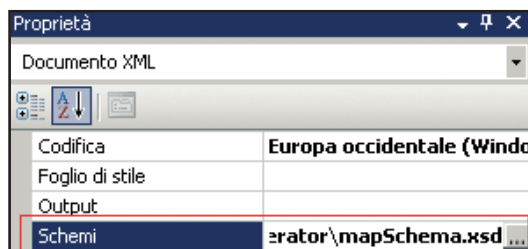



Figura 4: ***Proprietà Schema del file XML***

CREAZIONE DI UN WRITER

Siamo così arrivati al punto centrale della nostra applicazione, utilizzare le informazioni derivanti dal mapping effettuato dall'utente e dagli schemi recuperati dal database per generare il codice vero e proprio.
Riprendiamo allora, dal nostro primo articolo, lo scheletro della classe che gestisce il processo di generazione:

```
<Guid("9695a0c9-7a6a-49d6-b697-590cd6177bba")> _
Public Class MioGeneratore
    Inherits BaseCodeGeneratorWithSite

    Protected Overrides Function GenerateCode(ByVal
        inputFileName As String, ByVal inputFileContent As
                                        String) As Byte()

    End Function
End Class
```

Notiamo subito un limite : la funzione Generate Code prende come parametri il nome ed il contenuto del file di input editato dall'utente e restituisce una matrice di Byte, Visual Studio poi utilizzerà questa matrice per scrivere il file che avrà lo stesso nome del file di input e l'estensione dei file di codice del progetto. Ad esempio, se siamo in un progetto VB e il file di input si chiama input.xml verrà generato, nella stessa directory, il file input.vb con il codice.
A noi però occorre generare più di un file, almeno due : uno, quello primario, che contiene la descrizione in XML del mapping (così come viene fuori dal merge delle informazioni fornite dall'utente e quelle recuperate dal database) e un altro che contiene un file (.vb o .cs) con il codice generato.
Occorre quindi superare la limitazione del rapporto 1:1 tra input e output.
Per realizzare questo obiettivo stabiliamo che il Writer non restituisca direttamente un solo flusso di Byte, ma piuttosto una matrice di oggetti OutputResult che hanno due sole proprietà: la matrice di Byte da scrivere e il nome del file in cui deve essere scritta.
In questo modo facciamo sì che in fase di chiamata della funzione GenerateCode possano essere scritti più file distinti :

```
Function GenerateCode(ByVal inputFileName As
    String, ByVal inputFileContent As String) As Byte()
   Dim defaultOutput As Byte() = Nothing
   Dim parser As New InputParser
   Dim codeWriter As New VbCodeWriter(ProjectPath,
                                    inputFileName)
   Dim results As List(Of OutputResult) =
         parser.GetResult(inputFileContent, codeWriter)
      For Each out As OutputResult In results
         If String.IsNullOrEmpty(out.Filename) Then
' il nome del file non esiste quindi i Bytes
                        rappresentano il file principale
            defaultOutput = out.Content
         Else
' il nome del file esiste: i Bytes sono scritti nel file
                                                relativo
            WriteAllBytes(out.Filename, out.Content,
                                                  False)
         End If
      Next
   Return defaultOutput
End Function
```

I TEMPLATE

Resta da vedere come avviene la generazione vera e propria del codice. Già nel primo articolo accennavamo alla possibilità di utilizzare CodeDom per costruire un oggetto che rappresenti una sorta di meta-programma e che possa essere poi volta a volta "tradotto" in codice di un dato linguaggio da un CodeDomProvider specifico.
Tuttavia, pensate soltanto all'esempio, che abbiamo visto, su come generare il meta-codice che rappresenta un semplicissima classe con una proprietà :

```
Imports System.CodeDom
Public Class CodeDomTest
   Private Shared Function CreateCodeNamespace() As
                                          CodeNamespace
      Dim ns As New CodeNamespace("Prova")
      Dim cls As New CodeTypeDeclaration("Test")
      cls.IsClass = True
      Dim f As New
   CodeMemberField(GetType(Integer).FullName, "_ID")
      cls.Members.Add(f)
      Dim m As New CodeMemberProperty
      m.Name = "ID"
      m.HasGet = True
      m.HasSet = True
      m.Type = New
         CodeTypeReference(GetType(Integer).FullName)
      Dim privateFieldRef As New
                     CodeFieldReferenceExpression(New
                  CodeThisReferenceExpression(), "_ID")
      Dim ReturnExpr As New
            CodeMethodReturnStatement(privateFieldRef)
      m.GetStatements.Add(ReturnExpr)

      Dim rightExpr As New
               CodePropertySetValueReferenceExpression
      Dim setExpression As New
         CodeAssignStatement(privateFieldRef, rightExpr)
      m.SetStatements.Add(setExpression)
      cls.Members.Add(m)
      ns.Types.Add(cls)
      Return ns
```

```
    End Function
End Class
```

bene, pensate che tutto questo serve a produrre solo questo semplicissimo codice:

```
Namespace Prova
    Public Class Test
        Private _ID As Integer
        Public Property ID() As Integer
            Get
                Return _ID
            End Get
            Set(ByVal Value As Integer)
                _ID = Value
            End Set
        End Property
    End Class
End Namespace
```

ve lo immaginate il lavoro necessario a produrre, con questo sistema, una serie di classi complesse? No, evidentemente la cosa non è proponibile! Ed allora dobbiamo trovare un'altra strada.
In definitiva cosa abbiamo? Da una parte i nomi delle tabelle, delle colonne (con i relativi tipi di dati), quindi in definitiva una lista di stringhe, dall'altra molto testo che rappresenta il codice del programma. Prendiamo, ad esempio, la colonna CustomerID della tabella Customers ; essa verrà tradotta in una proprietà della classe Customers in questo modo:

```
    Private _CustomerID As System.String
    Public Property CustomerID() As System.String
        Get
            Return _CustomerID
        End Get
        Private Set(ByVal Value As System.String)
            _CustomerID = Value
    End Set
    End Property
```

la stessa cosa avverrà per tutte le altre colonne; quello che cambia, è solo il nome della colonna/proprietà e il nome del tipo di dati.
Da ciò è facile ricavare un modello dove, tra i simboli "{" e "}", possiamo indicare le variabili:

```
    Private _{name} As {type}
    Public Property {name}() As {type}
        Get
            Return _{name}
        End Get
        Set(ByVal Value As {type})
            _{name} = Value
            Me.Values("{name}") = Value
        End Set
    End Property
```

Abbiamo quindi creato una classe Template che dispone di vari metodi che, sostanzialmente, hanno come parametri, da una parte un Dictionary con coppie nomi/valori e dall'altra una stringa contenente il modello. I metodi in questione :

- analizzano il modello con le regular expressions per ricavare i nomi delle variabili
- sostituiscono ai segnaposti i valori presi dal Dictionary
- restituiscono la stringa risultato

il funzionamento è pressappoco questo:

```
Dim args As New NameValueCollection()
args("name") = "CustomerID"
args("type") = "System.String"
Dim modello As String = "una stringa modello"
Return Template.Transform ( args, modello)
```

In questo modo possiamo creare tutto il nostro codice nei Templates ed unirli, in fase di generazione, ai dati.

GENERAZIONE MULTILINGUAGGIO

Se i template sono comodi per elaborare i metadati, c'è però il problema che sono legati ad un linguaggio, se vogliamo far sì che il nostro generatore funzioni sia in VB che in C# dovremmo gestire templates diversi per ciascun linguaggio, con inevitabili problemi di disallineamento tra le due versioni.
Molti avranno sicuramente sentito parlare dell'ottimo IDE SharpDevelop, un clone Open Source di Visual Studio, una delle *features* più entusiasmanti di questo prodotto è la possibilità di convertire il codice da VB a C# e viceversa.
Questa funzione è resa possibile da una libreria a corredo di SharpDevelop chiamata *ICSharpCode.NRefactory.Dll.*
Questa libreria funziona anche con Visual Studio, quindi si potrebbe referenziarla nel progetto del nostro Custom Tool e poi utilizzarne le funzioni di conversione.

USARE IL CODEDOM

Il CodeDom è la via maestra per produrre generatori compatibili con tutti i linguaggi. Purtroppo, trattandosi di astrarre i costrutti di programmazione, non è proprio la cosa più agevole da gestire.

**Alcuni riferimenti si possono trovare in:
http://www.devx.com/dotnet/Article/15678/0/page/1
http://www.codeproject.com/dotnet/CompilingWithCodeDom.asp
http://aspalliance.com/1009_Using_CodeDOM_in_NET_20**

L'INSTALLAZIONE

Una volta terminato il nostro Custom Tool avremmo quindi due classi, una per VB e una per C#, che la DLL espone a COM, referenziate per GUID :

```
<Guid("db991a41-cdbb-4d63-9caa-c4e54d70e1e6")> _
Public Class VBMappingGenerator ...

<Guid("72cd9213-b889-4522-b8ce-12b9f63d95a1")> _
Public Class CSMappingGenerator ...
```

Compiliamo tutto il progetto (Visual Studio provvede automaticamente alla registrazione delle classi COM) e, come abbiamo visto nel nostro primo articolo, registriamo i nostri generatori nel Registry di Windows creando le chiavi :

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VisualStudio\8.0\Generators\{164b10b9-b200-11d0-8c61-00a0c91e29d5}\DBMappingGenerator per Visual Basic e

∑

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VisualStudio\8.0\Generators\{fae04ec1-301f-11d3-bf4b-00c04f79efbc}\DBMappingGenerator per C#

Nella prima chiave inseriamo un valore stringa chiamato CLSID corrispondente a "db991a41-cdbb-4d63-9caa-c4e54d70e1e6" corrispondente alla GUID di VBMappingGenerator.

Nella seconda chiave inseriamo un valore stringa chiamato CLSID corrispondente a "72cd9213-b889-4522-b8ce-12b9f63d95a1" corrispondente alla GUID di CSMappingGenerator.

In entrambe le chiavi dovremo poi inserire un valore di tipo DWORD chiamato GeneratesDesignTime Source corrispondente a 1.

A questo punto siamo già in grado di utilizzare i generatori.

IL GENERATORE IN AZIONE

In un progetto, VB o C#, inseriamo un file XML (io consiglierei comunque di cambiargli estensione, da .xml a .map, per riconoscerlo meglio) e, nelle proprietà, alla voce *Strumento personalizzato* (o *Custom Tool* se Visual Studio è in inglese) scriviamo "DBMappingGenerator".

Se le cose sono andate a buon fine, a questo punto modificando il file con cui mappiamo il database verranno generati:

- un file XML con la struttura completa del database
- un file .vb o .cs (a seconda del tipo progetto) con il codice

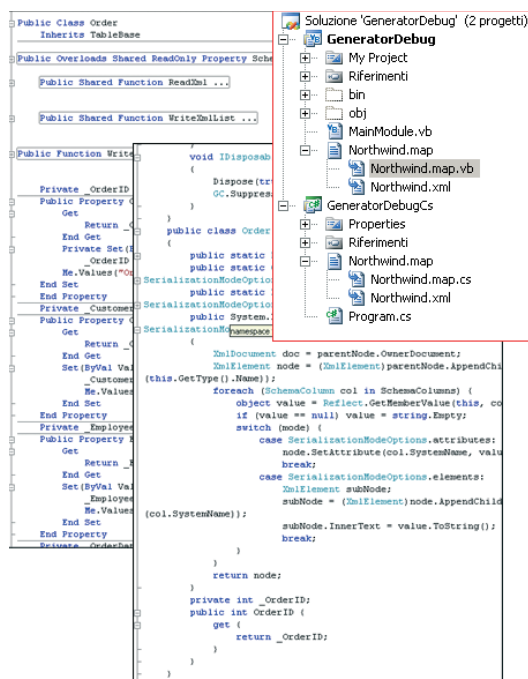



Figura 5: Il generatore in azione

CONVERSIONE CON ICSHARPCODE.NREFACTORY

Noi abbiamo scritto il codice dei templates in VB quindi, per C#, si potrebbe creare un writer specifico che richiama il codice generato per VB e lo converte con ICSharpCode.NRefactory.
Naturalmente è possibile fare anche il contrario: scrivere il generatore primario in C# e convertire il codice in VB.
Purtroppo la conversione non è mai precisa al 100%, l'unica soluzione veramente sicura sarebbe quindi utilizzare il CodeDom, naturalmente se ne vale la pena (ad esempio se si sviluppasse un Custom Tool commerciale).
Nell'esempio allegato ad CD, per ragioni di tempo, non abbiamo effettuato l'elaborazione di codice con il CodeDom e la conversione con ICSharpCode.NRefactory si è rilevata troppo imprecisa per essere utilizzabile, quindi il generatore per C# produce attualmente solo il documento di mapping XML, riteniamo tuttavia che il codice proposto sia utile anche ai programmatori C# come base per elaborare un proprio generatore.

SPECIFICHE DEL FILE DI MAPPING

Il file di mapping che possiamo editare (quello che fa da guida al generatore) prevede alcuni semplici elementi.

<**database**> (obbligatorio) è l'elemento di primo livello che comprende gli attributi:
name – nome del database
connectionString – stringa di connessione (è quella usata in fase di sviluppo e non comparirà nel codice)
type – tipo del provider a scelta tra : SqlClient, Oledb e Odbc
namespace – Spazio di nomi da utilizzare nella

generazione
<**class**>(facoltativo) se presente, sotto <database>, rappresenta le opzioni di generazione:
generate – indica se generare o meno il codice
path – permette di definire un percorso di generazione diverso da quello predefinito

<**xslGenerator**>(facoltativo) se presente, sotto <database>, consente di trasformare il file XML di mapping generato (quello con il dettaglio di tutti i campi, non quello editato da noi) utilizzando un foglio di stile, gli attributi sono:
xslDocument - Path relativa del documento Xsl usato per la generazione
outputPath - Path relativa del risultato della trasformazione Xsl

<**table**> (facoltativo) se presente, sotto <database>, consente di definire il mapping di una tabella o vista, gli attributi possono essere :
name – nome della tabella
sysname – nome della classe generata corrispondente alla tabella
readOnly – se la tabella supporta o meno INSERT,UPDATE o DELETE (potrebbe non supportarli nel caso di una vista oppure quando non si vogliono generare i metodi relativi)
mappingType – auto se vogliamo che la mappatura dei campi avvenga in automatico (salvo gli override esplicitamente definiti negli elementi *field*) o explicit se vogliamo indicare noi i campi da inserire (definiti con gli elementi *field*)

<**field**>(facoltativo) se presente, sotto <table>, consente di definire il mapping di un campo della tabella con gli attributi:
name – nome del campo
sysname – nome della proprietà generata corrispondente al campo
keyField – se il campo è un campo chiave (indipendentemente se sia definito tale nel database)
remove – se non vogliamo che per il campo venga generata una proprietà (nel caso in cui *mappingType* di <table> sia *auto*.

<**object**>(facoltativo) se presente, sotto <table>, consente di definire un oggetto, nella classe corrispondente alla tabella, che proviene da un'altra tabella in rapporto 1:1, l'elemento ha come attributi:
ref – nome della tabella, che deve essere anch'essa definita, al quale si collega quella corrente
sysname – nome della proprietà che rappresenta l'oggetto
cascadeDelete – se devono essere generate le funzioni di cancellazione a cascata
cascadeUpdate - se devono essere generate le funzioni di aggiornamento a cascata
l' elemento ha come sottoelemento obbligatorio <relation> (possono essere definite più relation).

<**list**>(facoltativo) se presente, sotto <table>, consente di definire una collection di oggetti, nella classe corrispondente alla tabella, che proviene da un'altra tabella in rapporto 1:n, l'elemento ha come attributi:
ref – nome della tabella, che deve essere anch'essa definita, al quale si collega quella corrente
sysname – nome della proprietà che rappresenta la collection
cascadeDelete – se devono essere generate le funzioni di cancellazione a cascata
cascadeUpdate - se devono essere generate le funzioni di aggiornamento a cascata
l' elemento ha come sottoelemento obbligatorio <relation>(possono essere definite più relation).

<**relation**> (obbligatorio), presente sotto <object> e sotto <list>, consente di definire la relazione tra tabelle madre e figlia con gli attributi:
parentKey – nome di un campo chiave della tabella madre
childKey - nome di un campo chiave della tabella figlia

<**procedure**> (facoltativo) - se presente, sotto <database>, consente di definire il mapping di una stored procedure mappando automaticamente anche i parametri, come attributi ha :
name – nome della Stored Procedure
sysname – nome della classe corrispondente generata

<**resultTable**> (facoltativo)- se presente, sotto <procedure>, consente di definire una tabella restituita dalla stored procedure, gli attributi sono:
name – nome attributo alla tabella risultato
sysname – nome della classe corrispondente generata

SVILUPPO DEL PROGETTO

Il progetto di esempio per VB è già abbastanza testato e funzionante, comunque l'autore sarà lieto di ricevere consigli, proposte e collaborazioni per estenderne le funzionalità.
Francesco Smelzo può essere contattato all'e-mail francesco@smelzo.it

<**field**> (obbligatorio)- diverso da <field> che si trova sotto <table> quando si trova sotto <resultTable> presenta come attributi:
name – nome attributo alla tabella risultato
sysname – nome della classe corrispondente generata
type – tipo di dati

UTILIZZO

Una volta che il generatore avrà fatto il suo lavoro (in pratica ogni volta che modifichiamo il file di mapping), avremo a disposizione delle classi fortemente tipizzate come quelle, ad esempio, in **figura 6**.

Order
Class
→ TableBase
Campi
Proprietà
CustomerID
EmployeeID
Freight
OrderDate
OrderID
OrderSelection
RequiredDate
SchemaColumns
ShipAddress
ShipCity
ShipCountry
ShipName
ShippedDate
ShipPostalCode
ShipRegion
ShipVia
Metodi
Create (oltre a 1...
Delete
DeleteExpression
DeleteMe
GetSelection
InsertInto
InsertIntoExpress...
ReadXml
Select (oltre a 3...
SelectById
SelectCount (oltr...
SelectPage (oltre...
SelectTable (oltre...
Update
UpdateExpression
WriteXml
WriteXmlList
Tipi nidificati

CustomerObject

Customer
Class
→ TableBase
Campi
Proprietà
Address
City
CompanyName
ContactName
ContactTitle
Country
CustomerID
CustomerSelection
Fax
Phone
PostalCode
Region
SchemaColumns
Metodi
Tipi nidificati

Details

OrderDetail
Class
→ TableBase
Campi
Proprietà
Discount
OrderDetailSelect...
OrderID
ProductID
Quantity
SchemaColumns
UnitPrice
Metodi
Tipi nidificati

Figura 6: Le classi generate

Ogni operazione, a questo punto, può essere effettuata grazie ai metodi statici della classe che rappresenta la tabella.
Ad esempio, per invocare una selezione sulla tabella ordini sarà sufficiente:

```
Public Sub SelezionaOrdini()
    Dim where As New Order.OrderSelectionClass
  Dim man = New Manager("Data
        Source=Localhost;InitialCatalog=Northwind;
                        Integrated Security=True")
where.OrderIDSelection.Set(SelectionOperators.Lt,
                                             10250)
    Dim orders As List(Of Order) =
                                      Order.Select(man,
                                              where, "")

    For Each o As Order In orders
        Dim cust As Customer =
                             o.CustomerObject(man)
        Dim company As String = ""
        If cust IsNot Nothing Then
            company = cust.CompanyName
        End If
        Console.WriteLine("Ordine {0} da {1} in
                                                  data
                        {2}", o.OrderID, company,
                        o.OrderDate.ToShortDateString)
        Console.WriteLine(StrDup(20, "="))
        For Each od As OrderDetail In
                                          o.Details(man)
            Console.WriteLine(">ProductID:" &
                                        od.ProductID)
            Console.WriteLine(">Quantity:" &
                                         od.Quantity)
            Console.WriteLine(">UnitPrice:" &
                                        od.UnitPrice)
            Console.WriteLine()
        Next
    Next
End Sub
```

Una volta definita una stringa di connessione, si svolge tutto con le funzioni generate, senza richiamare il codice SQL sottostante.

CONCLUSIONI

Quello che abbiamo mostrato, ancorché funzionante, è solo un esempio delle potenzialità offerte dai generatori di codice, esempio che ognuno potrà sicuramente adattare alle proprie esigenze per ottenere un mapping del database rispondente al 100% alle proprie necessità.

Francesco Smelzo

I trucchi del mestiere

Tips & Tricks

Questa rubrica raccoglie trucchi e piccoli pezzi di codice, frutto dell'esperienza di chi programma, che solitamente non trovano posto nei manuali. Alcuni di essi sono proposti dalla redazione, altri provengono da una ricerca su Internet, altri ancora ci giungono dai lettori. Chi volesse contribuire, potrà inviare i suoi Tips&Tricks preferiti. Una volta selezionati, saranno pubblicati nella rubrica. Il codice completo dei tips è presente nel CD allegato nella directory *tips*\\ o sul Web all'indirizzo: cdrom.ioprogrammo.it.

C#

CALCOLARE LE DATE

Come posso calcolare le date delle Pasqua?

```
//Domenica di Pasqua
public static void EasterSunday
                    (int year, ref int month, ref int day)
{
    int g = year % 19;
    int c = year / 100;
    int h = h = (c - (int)(c / 4) - (int)((8 * c + 13) / 25)
                    + 19 * g + 15) % 30;
    int i = h - (int)(h / 28) * (1 - (int)(h / 28) *
            (int)(29 / (h + 1)) * (int)((21 - g) / 11));

    day   = i - ((year + (int)(year / 4) +
             i + 2 - c + (int)(c / 4)) % 7) + 28;
    month = 3;

    if (day > 31)
    {
        month++;
        day -= 31;
    }
}
```

E il suo overload

```
public static DateTime EasterSunday(int year)
{
    int month = 0;
    int day = 0;
    EasterSunday(year, out month, out day);

    return new DateTime(year, month, day);
}
```

CONVERTIRE I FORMATI

Come posso convertire una data dal formato americano a quello europeo?

```
string old_date = DateTimePicker1.Text.ToString();
string[] new_date = old_date.Split(new Char[] { '/' });
string eu_date = new_date[1] + "-" + new_date[0] + "-" +
                                              new_date[2];
```

OTTENERE UNA STRINGA DA UNA DATA

Come posso convertire una stringa in formato Datetime?

```
// String to DateTime
 String MyString;
 MyString = "1999-09-01 21:34 PM";

 DateTime MyDateTime;
 MyDateTime = new DateTime();
 MyDateTime = DateTime.ParseExact(MyString, "yyyy-MM
                                     -dd HH:mm tt",null);
```

OTTENERE UNA DATA DA UNA STRINGA

Come posso convertire un Datetime in una stringa?
//DateTime to String

```
 MyDateTime = new DateTime(1999, 09, 01, 21, 34, 00);
 String MyString;
 MyString = MyDateTime.ToString("yyyy-MM-dd HH:mm tt");
```

VB.NET

SVUOTARE IL CESTINO

Come posso svuotare "il cestino" in modo programmatico?

```
Dim Action As New ShellActions
Action.EmptyRecycleBin()

Public Class ShellActions
```

```
Shared Function _
SHEmptyRecycleBin(ByVal hWnd As Integer, ByVal pszRootPath
                                                  As String, _
ByVal dwFlags As Integer) As Integer
End Function

Sub New()
EmptyRecycleBin()
End Sub

Sub EmptyRecycleBin(Optional ByVal rootPath As String = "", _
Optional ByVal noConfirmation As Boolean = True,
                                    Optional ByVal NoProgress _
As Boolean = True, Optional ByVal NoSound As Boolean = True)

Const SHERB_NOCONFIRMATION = &H1
Const SHERB_NOPROGRESSUI = &H2
Const SHERB_NOSOUND = &H4

If rootPath.Length > 0 AndAlso rootPath.Substring(1, 2)
                                                  <> ":\" Then
rootPath = rootPath.Substring(0, 1) & ":\"
End If

Dim flags As Integer = (noConfirmation And
                                   SHERB_NOCONFIRMATION) Or _
(NoProgress And SHERB_NOPROGRESSUI) Or (NoSound And
                                               SHERB_NOSOUND)
SHEmptyRecycleBin(0, rootPath, flags)
End Sub
```

JAVA

CONTROLLARE L'ESISTENZA DI UN FILE

Come posso sapere se un file esiste?

```
mport java.io.File;
public class DetermineFileDirExists {
    private static void doTest() {
        // Create a File object
        File file1 = new File("README_InputFile.txt");
        File file2 = new File("BlaBlaBla.txt");
        boolean b = file1.exists();
        System.out.println();
        System.out.println("Does File/Dir " + file1 + " exist?
                                              (" + b + ")\n");
        b = file2.exists();
        System.out.println();
        System.out.println("Does File/Dir " + file2 + " exist?
                                              (" + b + ")\n");
    }
    /**
     * Sole entry point to the class and application.
     * @param args Array of String arguments.
     */
    public static void main(String[] args) {
        doTest();
    }
}
```

TROVARE LA DIMENSIONE

Come posso determinare la dimensione di un file?

```
import java.io.File;
import java.io.IOException;
public class SizeOfFile {
    private static void doCheckSize() {
        // Create a File object
        File file = new File("README_InputFile.txt");
        // Get the number of bytes in the file
        long fileLength = file.length();
        System.out.println(
                "The length (in bytes) of file " + file +
                " is " + fileLength + ".\n");
    }
    public static void main(String[] args) {
        doCheckSize();
    }
}
```

PHP

PHP INCONTRA JAVA

Come posso integrare PHP e JAVA?

È necessario abilitare l'estensione php_java.dll e avere una JVM installata. L'integrazione è poi possibile con un codice simile a quello che segue:

```
<?php
// get instance of Java class java.lang.System in PHP
$system = new Java('java.lang.System');
// demonstrate property access
echo 'Java version=' . $system->getProperty
                               ('java.version') . '<br />';
echo 'Java vendor=' . $system->getProperty
                                ('java.vendor') . '<br />';
echo 'OS=' . $system->getProperty('os.name') . ' ' .
         $system->getProperty('os.version') . ' on ' .
         $system->getProperty('os.arch') . ' <br />';
// java.util.Date example
$formatter = new Java('java.text.SimpleDateFormat',
                 "EEEE, MMMM dd, yyyy 'at' h:mm:ss a
                                                  zzzz");
echo $formatter->format(new Java('java.util.Date'));
?>
```

L'OGGETTO NULLAFACENTE

UNO DEGLI ERRORI PIÙ COMUNI NEI PROGRAMMI A OGGETTI È IL FAMIGERATO "RIFERIMENTO NULLO". POCHI LO SANNO, MA ESISTE UN MODO ELEGANTE PER LIBERARSENE UNA VOLTA PER TUTTE. SCOPRIAMOLO CON UN ESEMPIO IN RUBY

Ci risiamo: sono bastate poche ore perché il nostro nuovo sistema di reportistica esplodesse in una fiammeggiante spirale di bug. I sistemisti continuano a riavviare il server, ma dopo pochi minuti il sistema va nuovamente in crash. Eppure eravamo convinti di aver testato il codice riga per riga.
La situazione non sarebbe nemmeno troppo grave se questo fosse un normale periodo di lavoro – ma ovviamente non è così. L'intero dipartimento aziendale che usa il programma sta preparando un documento importante che deve essere pronto entro domani, e ha bisogno del nostro sistema per avere dati aggiornati all'ultimo minuto. (Così ci siamo ritrovati dozzine di utenti urlanti e armati di forconi davanti alla finestra dell'ufficio). Dobbiamo trovare il bug e correggerlo entro questa mattina.
Per fortuna il messaggio di errore è abbastanza chiaro:

```
[...] undefined method `format_for_printing' for
                         nil:NilClass (NoMethodError)
```

Il sistema è scritto in Ruby, e in questo linguaggio il valore nil significa "oggetto non inizializzato". L'eccezione arriva dal metodo *show_report*:

```
def show_report(r)
  puts "Report andamento di borsa:"
  puts r.format_for_printing
  puts Time.now
end
```

Il metodo riceve il report e mostra un'intestazione, seguita dalla stampa del report e dalla data (il metodo puts stampa sullo standard output). È chiaro che per qualche motivo stiamo passando un oggetto nullo a questo metodo, quindi il programma va in crash quando show_report chiama *format_for_printing* sull'oggetto. Questa situazione è tristemente nota ai programmatori di qualsiasi linguaggio a oggetti. Gli sviluppatori Java, in particolare, sono abituati ad avere a che fare con le famigerate *NullPointerException* – l'eccezione lanciata dal linguaggio quando si cerca di de-referenziare un riferimento nullo.
Un modo per evitare questa situazione ci sarebbe: non avere mai riferimenti nulli in giro per il codice. Il problema è che il valore nullo è utile in molti casi. In questo caso, ad esempio, ci basta esplorare il sistema per scoprire che il report nullo arriva direttamente dal modulo di aggiornamento. Il report viene scaricato in formato XML da Internet e convertito in un oggetto Ruby di classe *Report*. Ma il sistema remoto invalida periodicamente il report (che a questo punto contiene dati obsoleti che non devono più essere mostrati a nessuno) e ne genera uno nuovo. Durante questa operazione, che può durare parecchi minuti, il collegamento è inattivo. Il nostro sistema di aggiornamento non riesce a connettersi, segnala la cosa sul log e, non sapendo cos'altro fare, restituisce un oggetto nullo. La cosa ha senso, perché il nostro sistema deve continuare a funzionare senza lanciare eccezioni - ma purtroppo la conseguenza è che il sistema va in crash subito dopo, quando cerca di usare il report nullo.

Una possibile soluzione è quella di controllare che il report sia valido ogni volta che lo usiamo. Possiamo scrivere questo controllo nel codice che chiama *show_report*, o all'interno dello stesso *show_report*:

```
def show_report(r)
  return if r == nil
  puts "Report andamento di borsa:"
  puts r.format_for_printing
  puts Time.now
end
```

La prima riga del metodo controlla se l'argomento è nullo, e in questo caso esce subito (in Ruby l'istruzione if può essere messa dopo il codice da eseguire anziché prima; questo stile è molto usato quando il codice da eseguire consiste in

una sola riga, come in questo caso). Ora possiamo passare nil al metodo, e non succederà niente di male. Qualcuno chiama questo stile, in modo un po' dispregiativo, *programmazione* difensiva. Il problema della programmazione *difensiva* è che ci tocca affollare il codice di verifiche vagamente paranoiche per controllare che nessun parametro di nessun metodo sia nullo. Deve esserci un modo migliore per risolvere il problema.

IL PATTERN NULL OBJECT

Quando controlliamo che un riferimento non sia nullo, stiamo violando uno dei principi della programmazione a oggetti: "Tell, don't Ask" (ordina, non chiedere). Il principio dice: "Non guardare dentro un oggetto per decidere cosa farci. Invece, chiedi all'oggetto stesso di fare quello che deve, e sarà lui a decidere esattamente cosa". Il nostro codice, invece controlla che l'oggetto non sia nullo per decidere poi se chiamare o meno uno dei suoi metodi. Dovremmo sfruttare il polimorfismo e spostare parte del codice nel report. Ma come si può spostare del codice in un riferimento nullo? Dovremmo usare un oggetto al posto di nil. Ma allora, come facciamo ad inizializzare un oggetto per il quale non abbiamo dati validi?
Per cominciare, dobbiamo fare in modo che questo oggetto speciale possa ricevere gli stessi messaggi dei normali oggetti. Ad esempio, se il server che contiene il report non risponde, il sistema di aggiornamento può creare un oggetto che rappresenta un report nullo:

```
class NullReport
  def format_for_printing
    return "Report non valido - riprova tra pochi
                                          minuti"
  end
end
```

Un *NullReport* ha la stessa interfaccia di un normale report, quindi posso passarne uno a tutti i metodi che richiedono un report. A differenza del valore nil, un *NullReport* fallisce con grazia: quando gli chiedo di formattarsi per la stampa lui mi restituisce un messaggio esplicativo. Se il modulo di aggiornamento restituisse un *NullReport* anziché un nil, potremmo far passeggiare questo oggetto per il sistema senza timore di detonazioni:

```
report = NullReport.new
show_report(report)
```

Queste righe stampano un messaggio che invita l'utente ad aspettare e riprovare dopo un po'. Non è bellissimo da vedere, ma sicuramente meglio di un server che si autodistrugge ogni volta che un altro server non risponde:

```
Report andamento di borsa:
Report non valido - riprova tra pochi minuti
Tue Jan 30 21:49:29 +0100 2007
```

Questo è un esempio del pattern Null Object. Un Null Object è un oggetto che rappresenta il valore nullo per tutti gli oggetti del suo tipo. Tipicamente i suoi metodi non fanno niente, o restituiscono valori predefiniti.

UN NULL OBJECT IN QUATTRO RIGHE

Molti linguaggi dinamici permettono di scrivere un Null Object in modo ancora più semplice. In linguaggi come Ruby o Python possiamo definire un metodo "acchiappatutto". In Ruby questo metodo magico si chiama method_missing:

```
class NullObject
  def method_missing(*)
  end
end
```

Ogni volta che qualcuno chiama un metodo inesistente su un NullObject, Ruby instrada la chia-

UN PATTERN FUORI CATALOGO

A differenza della gran parte dei pattern usati comunemente, non troverete Null Object nel famoso libro Design Patterns. La sua prima formulazione "ufficiale" è stata scritta l'anno successivo da Bobby Woolf (HYPERLINK "http://www.cs.oberlin.edu/~jwalker/refs/woolf.ps"www.cs.oberlin.edu/~jwalker/refs/woolf.ps).

Il libro che ha reso popolare il Null Object è invece Refactoring, scritto da Martin Fowler nel 1999. Il libro, che è già un classico, insegna l'arte di trasformare il codice. Se vi siete mai trovati davanti ad un pezzo di codice troppo contorto per modificarlo serenamente, questo è il libro che fa per voi: contiene una serie di istruzioni passo-passo per ristrutturare e pulire il codice senza introdurre errori.

Negli ultimi anni il "refactoring" è diventato popolare, e oggi tutti i migliori IDE (Eclipse, Microsoft Visual Studio, IntelliJ IDEA) includono un menu che permette di applicare automaticamente molte delle tecniche elencate nel libro. Alcuni refactoring, però, richiedono un po' di intelligenza umana e non possono essere facilmente automatizzati. Uno tra questi è appunto Introduce Null Object, che trasforma un riferimento nullo in un oggetto.

mata verso il metodo speciale method_missing. L'asterisco al posto dei parametri significa semplicemente che il metodo può prendere qualsiasi lista di argomenti. Proviamolo:

```
report =  NullObject.new
show_report(report)
```

Il risultato è:

```
Report andamento di borsa:
nil
Tue Jan 30 21:52:47 +0100 2007
```

In questo caso il risultato non è molto bello da vedere. Il problema è che ciascun metodo in Ruby restituisce sempre un valore. Dato che la chiamata a *method_missing* non restituisce nulla di esplicito, Ruby restituisce automaticamente nil. Il risultato viene stampato sullo schermo, per lo sconcerto degli utenti. Quindi questa tecnica non è l'ideale in questa circostanza, ma può essere utile quando il client non utilizza il valore restituito dal "metodo nullo". Magia dei linguaggi dinamici. Ma non è finita: in Ruby esiste un modo ancora più bizzarro di applicare il pattern Null Object.

TIPI MORBIDI E TIPI DURI

Cosa cambia se vogliamo implementare il pattern Null Object in un linguaggio staticamente tipato, come C# o Java? Non molto. In Java, il metodo show_report avrebbe questo aspetto:

```
public static void showReport
                          (Report r) {
  System.out.println
       ("Report andamento di borsa:");
  r.formatForPrinting();
  System.out.println(new Date());
}
```

Il metodo accetta solo oggetti di tipo Report, quindi NullReport deve ereditare da Report. In generale, un Null Object in Java deve ereditare dalla classe dell'oggetto, o forse entrambi devono implementare la stessa interfaccia, o ereditare dalla stessa classe.

Nei linguaggi dinamici non esiste questo limite, quindi il Null Object può avere qualsiasi classe. L'importante è che abbia i metodi che servono ai suoi client.
Un esempio di Null Object in Java è nella libreria di crittografia, javax.crypto. Questo package include una classe Cipher, che si occupa di criptare e decriptare un testo. Questa classe ha anche una sottoclasse di nome NullCipher. In questo caso, quindi, il Null Object eredita direttamente dalla classe dell'OggettoReale.
Tutti i metodi che inizializzano un NullCipher sono vuoti. I metodi che si occupano di elaborare il testo, invece, restituiscono lo stesso argomento che gli viene passato: quindi se si passa un testo attraverso un NullCipher, il testo torna fuori così come è entrato. Se un pezzo di codice usa un Cipher, basta passargli un NullCipher per disabilitare la crittografia.

UN TRUCCO MENTALE JEDI

Guardate questo codice:

```
class NilClass
  def format_for_printing
    return "Report non valido"
  end
end

show_report(nil)
```

In Java, *null* è un valore speciale che identifica un riferimento senza un oggetto. In Ruby, invece, nil è un oggetto come gli altri – possiamo definirlo il Null Object di sistema. Provate a scrivere nil.class in un interprete Ruby, e scoprirete che la classe di *nil* si chiama *NilClass*. NilClass ha una sola istanza (è un Singleton), che è appunto nil. Inoltre in Ruby le classi non sono mai chiuse: posso sempre aprire una classe già definita e aggiungerle dei metodi. Quindi il codice che abbiamo appena visto aggiunge un metodo alla classe *NilClass*, e d'ora in poi questo metodo è disponibile sull'oggetto nil. Il risultato è:

```
Report andamento di borsa:
Report non valido
Tue Jan 30 21:58:43 +0100 2007
```

È chiaro che tanta potenza va usata con cautela: non è bello inquinare il sistema modificando il comportamento di un oggetto globalmente visibile e importante come nil. Ma questo codice vi dà l'idea di come un linguaggio dinamico vi permette di applicare i design pattern in modo creativo. E con questo invito a non abusare del vostro potere, vi diamo appuntamento al prossimo numero di ioProgrammo per esplorare un nuovo pattern.

Paolo Perrotta

JBEHAVE: NON CREDO AI MIEI OCCHI…

JBEHAVE RAPPRESENTA PER IL BEHAVIOUR-DRIVEN DEVELOPMENT L'ANALOGO DI JUNIT PER IL TEST-DRIVEN DEVELOPMENT. ECCO I PUNTI DI FORZA E LE IDEE INNOVATIVE PER PREFERIRLO AL RE INCONTRASTATO DEI FRAMEWORK PER I TEST

Lo sviluppo software ha portato, negli anni, a definire numerosi formalismi e paradigmi per gestire la complessità: partendo dallo sviluppo procedurale e modulare si è arrivati alla programmazione orientata ai progetti. Questo ha mitigato le difficoltà nella creazione e nella manutenzione di progetti complessi, ma non ha risolto un altro problema: il cambiamento, anche in corso d'opera, dei requisiti da parte dei clienti. Infatti queste variazioni hanno un impatto non solo nel design complessivo ma, e questo è l'aspetto più difficile da controllare, nella stabilità delle parti di codice già verificato.
Da sempre si è tentato di minimizzare questi cambiamenti imponendo modifiche minime ai requisiti. Negli ultimi anni si sono affacciate metodologie, cosidette "agili", che ribaltano completamente la situazione e anziché "frenare" il cambiamento, lo considerano parte integrante dello sviluppo e tentano di adottare soluzioni che si sono rivelate utili anche in altri contesti: lo sviluppo di test automatici pervasivi. Frutto di queste metodologie è il Test-Driven Development (oramai consolidato; chi volesse maggiori informazioni può consultare il box [1]). Da queste idee ne sono nate altre sempre più vicine al modo di pensare del cliente e adatte ai moderni progetti software…

DOMAIN-DRIVEN DEVELOPMENT

Eric Evans è il padre del cosidetto Domain-Driven Development (o DDD, [1]); tale modello si basa su una considerazione: qualunque problema reale (considerato nella sua interezza si usa parlare di "dominio del problema") per essere risolto ha bisogno di una sua rappresentazione formale (tale rappresentazione viene anche detta "modello") la quale, a sua volta, per essere realizzata deve essere implementata (a questo livello si parla di "codice").

Passando dal dominio al modello e, ancor più, al codice, la terminologia usata inizialmente (dagli esperti) va, via via, scostandosi sempre più fino a diventare irriconoscibile; è comune che guardando al codice nessuno possa comprendere il problema originario, almeno nei termini in cui si esprimono gli esperti. Questa situazione può verificarsi fin da subito, ma è comune che degeneri quando ci sono rifattorizzazioni: quasi mai vengono cambiati i nomi delle classi, dei metodi o dei singoli attributi. Questo diventa un problema sempre più ostico in quanto, con l'evolversi del progetto, anche il modello da cui deriva il codice viene raffinato; venuti meno i riferimenti terminologici, viene meno anche quel collegamento diretto che c'era in origine tra i due mondi, portando ad introdurre errori semantici sempre più difficili da risolvere (i diversi mondi "non si parlano" più). Ecco la necessità di mantenere un linguaggio comune sia nel descrivere il dominio di origine sia nel descrivere il modello e, in ultima analisi, il codice stesso. Qual è questo linguaggio? È quello degli esperti del dominio che usano nella loro vita di tutti i giorni. Ecco, per esempio, un codice autoesplicatico anche nel dominio di partenza (che è quello scolastico):

```
public Verbale scrutinio(Scuola qualeScuola,
                         Classe qualeClasse)
{
  Verbale verbale = new Verbale(qualeClasse);
  Alunno[] classe = Repository.getInstance().
           prendiAlunniPerClasse(qualeClasse);
       if (classe!=null)
           for(int i=0; i<classe.length; i++){
            Alunno alunno = classe[i];
            int quantiDebiti =
                      alunno.getNumeroDebiti();
            if (qualeScuola.getRegole().daBocciare(
                           quantiDebiti ) )
              verbale.boccia( alunno );
```

IL PROGETTO

JBehave non è un framework ancora stabile. Infatti la release provata è la 0.5.0. Esistono ancora molte caratteristiche non implementate e altre che vanno raffinate ulteriormente.
È possibile contribuire al progetto in quanto è distribuito in forma sorgente.
Fare riferimento al sito Web http://www.jbehave.org/, per ulteriori informazioni.

```
            else
              verbale.promuovi( alunno,
                                      quantiDebiti );
            }
    return verbale;
  }
```

ha senso per un qualsiasi insegnante in grado di leggere i costrutti base del linguaggio.
Guardando al codice di esempio, si ha evidenza che ogni scuola può avere regole diverse per la bocciatura, non c'è una dipendenza su dove sono memorizzati gli alunni (nessuna SELECT o apertura di file, tanto per intenderci) né istruzioni di visualizzazione dei risultati in un metodo che appartiene unicamente alla logica di dominio; questo perché si presuppone che l'applicazione sia implementata usando layer diversi. Secondo Evans ogni applicazione dovrebbe prevedere i seguenti layer: User Interface (o Layer di presentazione dei risultati), Application Layer (coordina le diverse attività dell'applicazione), Domain Layer (come si è visto con un linguaggio che deve rispecchiare quello del dominio di partenza) e Infrastructure Layer (supporto per gli altri layer; di solito qui si implementa lo strato di persistenza).
Se un domani ci dovesse essere un cambiamento nel dominio (per esempio, se non esistessero più i debiti ma si ritornasse agli esami di riparazione!) questi cambiamenti dovrebbero essere riportati sia nel modello sia nel codice. Pertanto la rifattorizzazione è un'attività sempre più frequente (ecco la necessità, come per la TDD, di scrivere dei test automatici che verifichino la stabilità del sistema dopo ogni rifattorizzazione).

BEHAVIOUR-DRIVEN DEVELOPMENT

Behaviour-Driven Development (BDD, http://behaviour-driven.org/) è una metodologia di sviluppo che si concentra sul comportamento atteso (behaviour) di un'applicazione (o parte di essa). Per ogni porzione di programma dovrebbe essere chiara la risposta ad una domanda del tipo "Cosa dovrebbe fare questa porzione di programma?". Riuscendo a formalizzare queste domande in test, si potrebbe adottare uno sviluppo di tipo Test-Driven Development, utilizzando al tempo stesso un linguaggio più consono al dominio applicativo, aderendo alle indicazioni del Domain-Driven Development. In particolare verrà mostrato l'uso di JBehave (sito di riferimento http://www.jbehave.org/, **Figura 1**), un framework Java per lo sviluppo di applicazioni secondo BDD.

Figura 1: La home page del progetto JBehave

DA JUNIT A JBEHAVE

JBehave si basa sulla libreria JUnit. Le classi che fanno parte dei test si basano su una convenzione sul nome: esso deve essere uguale alla classe da testare seguito dal suffisso "Behaviour" (comportamento). Ognuna di queste classi dovrebbe contenere tanti metodi quante sono i comportamenti da verificare: ogni metodo inizia con "should" (dovrebbe); ecco, per esempio (restando nel mondo della scuola), due possibili test per lo scrutinio (manca il metodo di utilità "classeConDebiti"; per i sorgenti completi far riferimento al progetto allegato alla rivista):

```
public class VerbaleBehaviour  {

  public void
        shouldTuttiPromossiQuandoNonHannoDebiti()
                                 throws Exception{
   Verbale verbale =  (new ScuolaAzioni()).
    scrutinio(new Scuola(), classeConDebiti( 0 ) );
   Ensure.that(
              verbale.getElencoBocciati().size()==0
                                                 );
  }

  public void
        shouldTuttiBocciatiQuandoHannoSoloDebiti()
                                 throws Exception{

   Scuola scuola = new Scuola();
   Classe classe = classeConDebiti(

      scuola.getNumeroMaterie() );
          Verbale verbale = (new ScuolaAzioni()).
     scrutinio(scuola,  classe);
```

```
  Ensure.that(
                verbale.getElencoBocciati().size()==
                         classe.getAlunni().length );
  }
}
```

Si noti che fin qui non c'è nulla di nuovo rispetto a JUnit se non la sostituzione di "test" con "Behaviour" nel nome della classe e "test" con "should"nel nome del metodo. Allo stesso modo c'è un'ulteriore sostituzione: anziché usare "assert", come si fa in JUnit, si usa "Ensure.That". Queste scelte non sono casuali: usare "Behaviour" indica qualcosa di più concreto di "Test"; dice che quello che è sotto analisi è il comportamento atteso, non altro. Usare "should" indica cosa ci si aspetta come comportamento relativo alla classe in sé, quindi specifico, e il comportamento da testare è uno e solo uno: ed esso è quello descritto dal resto del nome del metodo! Ma essendo "should" un condizionale, indica anche che "dovrebbe" essere così, ma che nulla è fisso e immobile. Se il test non ha più significato (magari in seguito ad un refactoring del codice), lo si può eliminare in quanto non è più di alcuna utilità.

ESEGUIRE I TEST

Non resta che mandare in esecuzione i test appena scritti. Per farlo si può aprire una console dei comandi e digitare il seguente comando:

```
>java -cp <fileJarBehave>;. jbehave.core.Run
         it.ioprogrammo.behaviour.VerbaleBehaviour
```

Se tutto va a buon fine, si ha la stampa del numero di test superati; altrimenti c'è la segnalazione di un'anomalia attraverso un'eccezione (si veda la **Figura 2**).

Questo comando dovrebbe essere eseguito per ogni test da verificare. C'è anche la possibilità di specificare un'unica classe che contiene i riferimenti a tutte le classi di test di un progetto e mandare in esecuzione solo questa; la classe deve implementare l'interfaccia jbehave.core.behaviour.Behaviours e, in particolare, il metodo getBehaviourClasses:

```
public class AllBehaviours implements Behaviours
{

 public Class[] getBehaviourClasses() {
  return new Class[] {

  VerbaleBehaviour.class,
   AltraClasseBehaviour.class,

  UlterioreClasseBehaviour.class,
 };
 }
}
```

JUNIT O JBEHAVE?

Benché queste indicazioni possano sembrare banali (in fondo c'è stato solo un cambio dei nomi usati) esse sono di grande utilità per i novizi di TDD. Sul sito http://dannorth.net/introducing-bdd/ è riportata l'esperienza di uno degli autori di JBehave; esperto di TDD, insegnava la metodologia in diversi corsi professionali; per sua esperienza, a molti degli studenti a cui insegnava TTD il solo cambiamento di nomenclatura ha portato ad un evidente vantaggio in termini di concretezza e indicazioni specifiche sulla scrittura dei test. Ma questa motivazione potrebbe essere troppo povera per abbandonare uno strumento consolidato come JUnit. Per fortuna JBehave non è solo questo!

USARE I TEST PER LA SPECIFICA DEI REQUISITI

I test scritti possono essere pensati come test scritti a posteriori per verificare un comportamento atteso, ma possono anche essere dei requisiti definiti in fase di analisi (e quindi

```
C:\WINDOWS\system32\cmd.exe

>java -cp lib/jbehave.jar;bin/ jbehave.core.Run it.ioprogrammo.behaviour.Verbale
Behaviour
.F
Time: 0.07s

Failures:

1) it.ioprogrammo.behaviour.Verbale should tutti bocciati quando hanno solo debi
ti [it.ioprogrammo.behaviour.VerbaleBehaviour]:
VerificationException: null: expected condition was not met:
        at jbehave.core.minimock.UsingConstraints.fail(UsingConstraints.java:245
)
        at jbehave.core.minimock.UsingConstraints.ensureThat(UsingConstraints.ja
va:233)
        at jbehave.core.Ensure.that(Ensure.java:71)
        at it.ioprogrammo.behaviour.VerbaleBehaviour.shouldTuttiBocciatiQuandoHa
nnoSoloDebiti(VerbaleBehaviour.java:20)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
        at java.lang.reflect.Method.invoke(Unknown Source)
        at jbehave.core.behaviour.BehaviourMethod.invoke(BehaviourMethod.java:72
)
        at jbehave.core.behaviour.BehaviourMethod.invoke(BehaviourMethod.java:57
)
        at jbehave.core.behaviour.BehaviourMethodVerifier.visitBehaviourMethod(B
ehaviourMethodVerifier.java:31)
        at jbehave.core.behaviour.BehaviourMethod.accept(BehaviourMethod.java:52
)
        at jbehave.core.behaviour.BehaviourClass.visitBehaviourMethods(Behaviour
Class.java:46)
        at jbehave.core.behaviour.BehaviourClass.accept(BehaviourClass.java:36)
        at jbehave.core.Run.runBehaviour(Run.java:49)
        at jbehave.core.Run.runBehaviourToStream(Run.java:25)
        at jbehave.core.Run.main(Run.java:18)

Total: 2. Failures: 1, Exceptions: 0.

>
```

Figura 2: Caso di test non andato a buon fine.

BIBLIOGRAFIA

[1] "Test-Driven Development by Example", K. Beck, Addison Wesley, 2003

[2] "Domain-Driven Design: Tackling Complexity in the Heart of Software", E. Evans, Addison-Wesley, 2003

non ancora realizzati). In questo caso si ricade nella situazione del TDD dove i test precedono la scrittura dell'analisi. Ancora una volta la nuova nomenclatura di JBehave è adatta (e, ancora, molto più di quella usata da JUnit), ma JBehave va ben oltre: definisce una serie di classi e strumenti di utilità che permettono, in maniera semplice, di scrivere i casi d'uso in file testuali e di generare automaticamente i test!

OGGETTI MOCK

JMock è una (delle tante) libreria Jave per realizzare oggetti Mock. Informazioni sulla libreria sono disponibili alla home del progetto: http://www.jmock.org/

SCENARI D'USO

Partendo dall'idea che un requisito possa essere formalizzato, quello che manca è riuscire a definire, nel codice, l'idea che uno scenario possa essere definito in linguaggio naturale come:

Given some initial context (the givens),
When an event occurs,
then ensure some outcomes.

Tradotto sarebbe come:

```
Scenario: Happy scenario
Given Account is in credit
When User requests cash
Then ATM should dispense cash
and ATM should return bank card
and account balance should be
reduced
```

```
addScenario(
 new ScenarioUsingMiniMock(
   "Happy scenario",
   getClass().getName(),
   new Givens(
     new AccountIsInCredit()
   ),
   new UserRequestsCash(),
   new Outcomes(
     new ATMShouldDispenseCash(),
     new ATMShouldReturnBankCardToCustomer(),
     new AccountBalanceShouldBeReduced()
   )
 )
);
```

Figura 3: Da una descrizione testuale ad un codice Java passando per JBehave

Dato un contesto
Quando accade un evento
Allora assicurati che avvenga

Indagando nei sorgenti di JBehave (pertanto non basta il dowload del solo JAR) si viene a conoscenza del fatto che questa caratteristica non è ancora realizzata completamente, ma esistono delle tracce seguendo le quali è possibile comprendere come sarà il risultato finale quando verrà rilasciata la versione stabile del framework. Nella cartella /core/analysis troviamo il file "user withdraws cash.txt"; il suo contenuto è:

```
Story: User withdraws cash

As a Bank card holder
I want to be able to withdraw cash from an ATM
So that I dont have to visit the bank

Scenario: Happy scenario
Given Account is in credit
When User requests cash
Then ATM should dispense cash
and ATM should return bank card
and account balance should be reduced
```

è riportata solo la prima parte (a seguire sono descritti altri scenari). Comunque possiamo notare che inizialmente viene definita una "story". Possiamo ritrovare questa parte in un file sorgente, e più precisamente nella classe example.stories.UserWithdrawsCash; tale classe estende jbehave.story.domain.Scenario Driven Story e presenta questo costruttore:

```
public UserWithdrawsCash() {
  super(
   new Narrative(
    "Bank card holder",
    "to be able to withdraw cash from an ATM",
    "I don't have to visit the bank"
  ),
   new AcceptanceCriteria()
 );

  addScenario(new ScenarioUsingMiniMock(
  "Happy scenario",
  getClass().getName(),
  new Givens(
   new AccountIsInCredit()
  ),
  new UserRequestsCash(),
  new Outcomes(
   new ATMShouldDispenseCash(),
   new ATMShouldReturnBankCardToCustomer(),
   new AccountBalanceShouldBeReduced()
  )
  )
 );
```

Come si può notare ogni frase che segue Givem, When e Then è stata trasformata in un metodo con l'usuale convenzione sulle lettere (maiuscole le iniziali di parole composte).
Indagando oltre sul codice si comprende che il costruttore usato per l'oggetto Narrative è:

```
Narrative(String role, String feature, String
                                        benefit)
```

Pertanto descrive, molto ad alto livello, lo scopo della classe da realizzare. Ancor più interessante è osservare come viene costruito (aggiunto) uno scenario: esso fa uso di ogget-

ti mock (grazie alla libreria JMock) che, come si vedrà nel seguito, permettono di simulare semplicemente degli oggetti complessi non ancora realizzati. Inoltre ciascuno scenario è composto da dei Givens (ovvero prerequisiti), come AccountIsInCredit (account con credito), delle azioni, come UserRequestsCash (l'utente richiede di prelevare soldi) e Outcomes, che in questo caso sono tre: ATMShould DispenseCash (i soldi dovrebbero essere dati dal bancomat), ATMShouldReturnBankCard ToCustomer (il bancomat dovrebbe restituire la carta usata) e AccountBalanceShould BeReduced (il conto corrente dovrebbe essere ridotto).
Com'è possibile notare, oltre ad essere autoesplicativo il nome di ogni classe usata è un semplicissimo modo di esprimere i requisiti per un'applicazione che, ancora, non esiste!
Non resta che implementare ogni singolo Given, Event ed Outcome. Ecco una classe per tipo (sempre tratta dagli esempi di JBehave). Per AccountIsInCredit:

```
public class AccountIsInCredit extends
                                GivenUsingMiniMock {
    public void setUp(World world) {
        Mock account = (Mock)
            world.get("account",
                        mock(Account.class));
account.stubs("getBalance").withNoArguments().
                        will(returnValue(50));
    }
}
```

Da osservare che viene estesa la classe GivenUsingMiniMock e viene definito un unico metodo (setUp). Tale metodo fa uso di Account.class che è la "classe" da testare. Ma attenzione: classe è stato indicato tra virgolette in quanto, a ben vedere, essa non è una classe, ma un'interfaccia che definisce unicamente i metodi da realizzare; ecco tale interfaccia:

```
public interface Account {
    int getBalance();
    int getOverdraftLimit();
}
```

Si presti attenzione alla potenza espressiva di una simile situazione: Account per ora è definita solo in termini di "funzionalità" da creare, e quindi da un'interfaccia e non certo da una classe concreta. Quando inizierà lo sviluppo verrà creata un'apposita classe che estende questa interfaccia. Quindi essa è pronta per prendere posto nei test della vecchia interfaccia! Ritorniamo alla classe AccountsIsInCredit; essa definisce anche il comportamento dell'oggetto creato:

```
account.stubs("getBalance").withNoArguments().
                        will(returnValue(50));
```

ovvero il metodo getBalance, senza argomenti, restituirà 50 (è un caso di conto corrente con del denaro; si è raggiunto lo scopo della classe; ora è usabile per modellare la situazione senza doverla implementare realmente, ma usando una classe mock, per l'appunto!).
La classe Event (come UserRequestsCash), in maniera simile, estenderà un EventUsingMini Mock e quella Outcome, come ATMShould DispenseCash, una classe OutcomeUsingMini Mock. Pertanto, a partire da uno scheletro autogenerato da un file testuale, si può procedere alla realizzazione dei test veri e propri, dapprima con oggetti mock, successivamente con le classi reali. Purtroppo, visto lo stato non avanzato del progetto, molto altro lavoro deve essere fatto in questa direzione, ma le idee ci sono e sono molto promettenti!

CONCLUSIONI

Testare il software non è un'operazione da demandare semplicemente alla buona volontà degli utenti. Non esiste un momento specifico per eseguire una sessione di debug, tuttavia è sempre opportuno fare precedere al rilascio di una versione Alpha una sessione di testing piuttosto intensiva. Questo vi garanirà nei momenti successivi di non dover spendere tempo nel risolvere segnalazioni non sempre precise.
Il Behaviour-Driven Development può essere senz'altro utilizzato in tutti quei contesti in cui la complessità dei problemi impone una rigida disciplina all'interno di tutto il team di sviluppo, a partire dagli analisti fino ai programmatori. Tale disciplina può anche essere presente in progetti più tradizionali che adottano il Test-Driven Development, ma con il BDD ne detta regole molto più rigide e definite. Come sempre ogni strumento o metodologia potrebbe essere ignorata dai "più bravi", ma è di supporto per tutti e indispensabile per i più. JBehave presenta notevoli punti di interesse e, in generale, è destinato a diventare lo strumento ideale a supporto di BDD. Non resta che attendere i suoi sviluppi e il rilascio di una release finalmente stabile.

Ivan Venuti

L'AUTORE

IVAN VENUTI

La passione per la programmazione lo ha sempre portato ad approfondire nuove tecnologie, tecniche di sviluppo e framework emergenti. Esperto di programmazione in ambiente J2EE, scrive abitualmente articoli per alcune tra le maggiori riviste di informatica italiane e ha al suo attivo numerosi libri sulla programmazione, in particolare su Java. Maggiori informazioni sul sito personale, raggiungibile all'indirizzo http://ivenuti.altervista.org.

SICUREZZA: QUANDO IL TESTO CONTA

LA CREAZIONE DI STRINGHE E L'ESTRAZIONE DI DATI IN C SONO UNA CONTINUA FONTE DI BUG CRITICI. IN C++, INVECE, ESISTONO SOLUZIONI LEGGIBILI E VERSATILI, SICURE RISPETTO AI TIPI E AGLI OVERFLOW. VEDIAMO QUALI.

Da questo numero ci dedicheremo ad un campo universalmente noto con il nome di "text processing". Illustreremo le tecniche per la manipolazione delle stringhe, per l'internazionalizzazione dei programmi, i vari algoritmi per effettuare modifiche e confronti, l'interpretazione e la validazione dell'input, e tanto altro ancora.
Si tratta di problemi che si presentano di continuo e in forme diverse al programmatore C++, che ha a sua disposizione un intero arsenale di tecniche per risolverli. Purtroppo, lo sviluppatore medio ne ha un quadro solitamente confuso e approssimativo, e ciò spiega l'inquietante epidemia di Patch all'ultimo minuto diffusa sul mercato. Pensate soltanto a tutte quelle falle di sicurezza la cui descrizione comprende le parole: "problema di buffer overflow", o: "SQL injection". In molti casi, andando a guardare le cause del problema, si scopre che i programmatori responsabili del bug hanno gestito le stringhe in maniera superficiale, o semplicemente scorretta.
In questa serie illustreremo alcune tecniche fondamentali, cercando di mantenere l'approccio ormai tipico di questa rubrica: dare rilievo a soluzioni robuste, che usino le reali potenzialità del C++, e che siano il più possibile indipendenti da specifici sistemi e compilatori.

IL PROBLEMA

In questa puntata ci occuperemo della creazione di stringhe, usando un esempio pratico. Ipotizziamo di essere nella (invidiabile) posizione del "programmatore di videogiochi". Abbiamo scritto una classe che gestisce la "Partita", e siamo arrivati quasi alla fine del nostro lavoro. Ci resta solo da definire la funzione *GameOver*, che si occupa di mostrare a video un messaggio di congratulazioni, di questo genere:

```
Complimenti, Tizio Caio!
Hai finito il gioco in 35,5 secondi,
con un punteggio di 250000!
```

GameOver ha, pertanto, due compiti distinti: costruire la stringa a partire dai dati della classe, e richiamare una qualche funzione che la mostri in una finestra di dialogo. Poiché ogni funzione deve avere una sola responsabilità, abbiamo strutturato la classe così.

```
class Partita {
public:
    void GameOver() {
        string messaggio(CreaStringaGameOver());
        MostraMessaggio(messaggio);
    }
    // […] resto dell'interfaccia […]
private:
    string CreaStringaGameOver();
    string nome_;
    long punteggio_;
    float secondi_;
};
```

Ora ci rimane solo da assolvere la prima responsabilità: costruire il messaggio in *CreaStringaGameOver*. Che metodo usereste, voi?

STRINGSTREAM

Non ho intenzione di lasciarvi in preda alla suspense fino all'ultimo paragrafo, quindi diciamolo subito: la via C++ per eseguire compiti di questo tipo è *stringstream*. La classe *stringstream* (un *typedef*, in realtà di *basic_stringstream<char>*) è definita nell'header <sstream>, e rappresenta l'astrazione di una stringa. Usandola è possibile inserire informazioni in una stringa trattandola come uno stream qualsiasi, per poi ottenerne una copia grazie alla funzione *str()*. Ecco l'implementazione di *CreaStringaGameOver*, ottenuta con stringstream.

```
string Partita::CreaStringaGameOver() {
    //dichiarazione
    stringstream messaggio;
    //inserimento
    messaggio << "Complimenti, "
```

```
        << nome_ << "!\n"
        << "Hai finito il gioco in "
        << secondi_ << " secondi"
        << " con un punteggio di "
        << punteggio_ << "!\n";

    //conversione a stringa
    return messaggio.str();
}
```

Questo esempio, per quanto semplice, mostra bene la procedura dichiarazione/inserimento/conversione da seguire per l'utilizzo di stringstream. Vediamo di analizzarne pregi e difetti.

I VANTAGGI DI STRINGSTREAM

1) **Stringstream evita il pericolo dei buffer overflow**, dal momento che la stringa alla quale lo stream fa riferimento viene ridimensionata automaticamente, e quindi non è richiesto al programmatore di costruire un buffer "tirando a indovinare" la sua dimensione massima.
2) **Stringstream è una soluzione sicura rispetto ai tipi**, dal momento che il tipo dei parametri concatenati non viene richiesto al programmatore, ma controllato subito e staticamente dal compilatore.
3) **Stringstream permette di concatenare molti tipi diversi**, attraverso una sintassi omogenea. Oltre a variabili e costanti di tipo primitivo, infatti, possono essere concatenati gli oggetti appartenenti a qualunque classe per la quale sia stato previsto un operatore di inserimento in stream. E, transitivamente, possono essere concatenati anche oggetti *implicitamente convertibili* in un tipo per il quale sia stato previsto un operatore di inserimento in stream.
4) **Stringstream fa pienamente parte della libreria standard**, e quindi oltre ad essere portabile e facile da mantenere, ha ottimi rapporti con il resto delle classi della STL. Se il framework sottostante decidesse, ad esempio, di supportare in una nuova versione i caratteri estesi, invece di quelli normali, dovremmo solo ritoccare i tipi in *wstring* e *wstringstream* (i typedef che stanno per basic_string<wchar_t> e basic_stringstream<wchar_t>), senza cambiare neanche una riga del resto del codice.
5) **Stringstream permette l'uso dei manipolatori standard**, consentendo così la formattazione della stringa in modo facilmente comprensibile.
6) **Stringstream può essere usato per estrarre informazioni da una stringa**, sfruttando il fatto che il comportamento di uno stream è bidirezionale. Conviene fare un esempio.

STRINGSTREAM PUÒ ESSERE USATO COME "ESTRATTORE"

Nuovo Problema. La finestra di dialogo introduttiva del gioco contiene dei campi testuali, il cui contenuto è di tipo *string*. In uno di questi richiedete l'anno di nascita del giocatore, che userete per assicurarvi che sia maggiorenne.
Cosa usereste per implementare una funzionalità del genere? Non c'è bisogno di cambiare: stringstream fa ancora al caso nostro.

```
int CalcolaEtà(const string& txtAnnoDiNascita) {
    stringstream tmp(txtAnnoDiNascita);
     int annoDiNascita;
    tmp >> annoDiNascita;
    return AnnoAttuale() - annoDiNascita;
}
```

Come vedete, stringstream può essere usato nelle due direzioni: potete estrarre informazioni da una stringa, processarle, e reinserirle – anche nello stesso stringstream di partenza, se volete.

STRINGSTREAM PERMETTE DI GESTIRE GLI ERRORI

Creare un gioco richiede anche nozioni di etologia, circa le abitudini comportamentali del Simpatico Giocatore. Posto di fronte ad un form che chiede informazioni personali, il Simpatico Giocatore sarà preso dall'impulso irresistibile di riempire tutte le voci con delle risposte a caso, quanto più dannose possibile per il parser che le leggerà.
C'è quindi da scommettere che *txtAnnoNascita* conterrà valori non-numerici, come "Ciao!", "Pippo", e via degenerando. È nostro preciso compito far sì che lo stato patologico del Simpatico Giocatore non si propaghi anche alla nostra applicazione.
Stringstream permette di sapere se un'operazione di inserimento o estrazione è andata a buon fine, interrogando lo stato dello stream.
Di questo stato sono significativi solo quattro bit, che corrispondono a delle costanti di tipo

IL BDW-GC E LE MACRO

Molte delle cosiddette "funzioni" che fanno parte dell'interfaccia C del garbage collector sono in realtà delle macro, che si espandono in modo differente a seconda delle impostazioni d'ambiente. GC_MALLOC, ad esempio, richiama la funzione GC_malloc in condizioni normali, e GC_malloc_debug nel caso in cui GC_DEBUG sia stato definito. La maggior parte delle macro del garbage collector segue il medesimo schema.

ios_base::iostate, come riportato in Tabella 1:

Costante	Funzione	Cosa significa se è impostato
goodbit	good()	Lo stream è a posto, e nessuno degli altri tre bit è impostato.
eofbit	eof()	Lo stream ha incontrato il carattere end-of-file.
failbit	fail()	Un'operazione di lettura o scrittura non è andata a buon fine.
badbit	bad()	È avvenuto un errore grave, e lo stream è in uno stato incoerente.

Ci sono diversi modi per sfruttare queste informazioni. Il metodo più semplice è utilizzare lo stream stesso come fosse un'espressione. Infatti, grazie all'operatore di conversione a *bool*, lo stream restituisce il valore di goodbit:

```
if (!(tmp >> annoDiNascita)) {
    //goodbit non è impostato!
    //l'estrazione non è andata a buon fine
}
```

Tuttavia, è un metodo che sconsiglio, perché porta a codice poco leggibile. Allungando il sorgente e richiamando una funzione esplicita si ottiene codice più facile da capire:

```
int CalcolaEtà(const string& txtAnnoDiNascita) {
    stringstream tmp(txtAnnoDiNascita);

    int annoDiNascita;
    tmp >> annoDiNascita;

    if (!tmp.good()) {
      throw(
        "Preghiamo il Simpatico Giocatore"
        " di consultare un buon analista";
      );
    }

    return AnnoAttuale() - annoDiNascita;
}
```

Questa procedura si rivela abbastanza prolissa, soprattutto quando si effettuano molte estrazioni successive. Per rimediare, si può chiedere allo stream stesso di generare un'eccezione automaticamente, in caso di errore nel parsing. La funzione **exceptions** (*stato*) permette di specificare in quale stato lo stream invocherà un'eccezione. Si possono indicare più stati, componendoli con l'operatore |. In questo esempio leggiamo tre numeri dallo stesso stream: se un inserimento fallisce (*failbit*), o lo stream incontra un errore indefinito (*badbit*), verrà lanciata un'eccezione di tipo **ios_base::failure**.

```
stringstream tmp(txtAnnoDiNascita);
tmp.exceptions(ios_base::failbit | ios_base::badbit);

try {
    long indirizzo, telefono, partitaIva;
    tmp >> indirizzo >> telefono >> partitaIva;
} catch(ios_base::failure&) {
    //gestisce l'eccezione in caso di errore nel parsing
    CreaFinestraDiDialogo("Sono stati inseriti dati
                                            errati");
}
```

GLI SVANTAGGI DI STRINGSTREAM

Come già detto, stringstream è la via maestra del programmatore C++. Questo non vuol dire che sia perfetta, ma semplicemente che ha molti pregi, e pochi difetti marginali. Eccoli.

1) **Stringstream costringe ad inserimenti a cascata poco leggibili**, che richiedono al programmatore una certa disciplina nella formattazione del codice, senza la quale è facile produrre sorgenti dall'aspetto orripilante.
2) **Stringstream non fornisce prestazioni ottimali in termini di velocità**, dal momento che costringe il programmatore ad usare almeno una variabile temporanea. Inoltre, usando memoria dinamica, può portare a riallocazioni interne. Non posso sottolineare questo punto, però, senza aggiungere che il 99% delle volte le preoccupazioni del programmatore C++ medio nei confronti della velocità sono frutto di un "complesso di ansia da prestazioni" dannoso e completamente ingiustificato.
3) **Stringstream costringe a mescolare il testo di formato con i dati da inserire nella stringa**, il che è una delle cause dei problemi di leggibilità, e rende molto difficile l'internazionalizzazione delle risorse.
4) **Stringstream è una soluzione prolissa**. Essere costretti a scrivere tutto un ciclo di

NOTA

"INSTALLARE" BOOST

Per usare format e lexical_cast, non serve alcuna "installazione" particolare di boost. Basta scaricare i sorgenti da www.boost.org, e impostare le directory di inclusione nel proprio IDE (se ne si sta usando uno).

"INSTALLARE" BOOST::LEXICAL_CAST E FORMAT

Per usare boost::lexical_cast e boost::format non è necessario collegare alcuna libreria. È sufficiente avere impostato boost fra le directory d'inclusione, e includere i file "<boost/lexical_cast.hpp>" e "<boost/format.hpp>", rispettivamente. Per evitare di dover specificare sempre boost:: davanti a lexical_cast e format, è bene dichiarare, da qualche parte prima dell'uso, using namespace boost;

dichiarazione/inserimento/conversione solo per trasformare un intero in una stringa è una lungaggine frustrante, che può essere evitata. Vediamo come.

"ATOI" E COMPAGNI NON SONO UNA SOLUZIONE

Vedendo usare stringstream come estrattore, un programmatore di derivazione C avrà forse pensato: "beh, posso fare lo stesso usando funzioni come *atoi*, senza dover dichiarare buffer o fare *cose strane*".

```
int CalcolaEtà(const string& txtAnnoDiNascita) {
    return AnnoAttuale() - atoi(txtAnnoDiNascita.c_str());
}
```

Il fatto è che le "cose strane" ***sono*** *il* C++, e quando si programma in C++ **l'approccio C diretto ha sempre gravi svantaggi**.
Senza sovraccaricamento, infatti, la libreria C è costretta a trattare *atoi*, *atol*, e *atof* come funzioni distinte, obbligando il programmatore finale a scegliere, e a cambiare le chiamate nel caso in cui cambi il tipo.
Allo stesso modo, le possibilità di conversione sono molto limitate, e non esiste una soluzione C generica e sicura. Tutte le soluzioni C per il passaggio di dati di tipo generico (type punning, ellissi, eccetera) sono raccapriccianti sotto il profilo della sicurezza rispetto ai tipi.
E che succede se atoi e compagni falliscono nella lettura? Si entra nel regno del comportamento indefinito. Anche con soluzioni C più "sicure" come *strtol* è difficile orizzontarsi in caso di errore.
Infine, le funzioni come atoi sono unidirezionali, e non permettono di passare, ad esempio, da un intero ad una stringa. Nonostante sia ancora usata da alcuni incorreggibili programmatori C, itoa non è mai esistita in nessuno standard.

LEXICAL_CAST

Nonostante tutto ciò, l'idea di avere una funzione "semplice e diretta" per le conversioni facili è ottima: va solo implementata in maniera C++. La libreria *boost* offre una soluzione molto interessante, scritta da Kevlin Henney: **lexical_cast**. Con un *lexical_cast* è possibile convertire un qualsiasi tipo Sorgente che definisca un operator<<(ostream&, Sorgente&), in qualsiasi tipo Destinazione che definisca un operator>> (istream&, Destinazione &), con la stessa sintassi degli operatori di conversione standard:

```
lexical_cast<Destinazione>(sorgente);
```

Così, grazie a lexical_cast, possiamo riscrivere l'esempio combinando la semplicità di atoi e la sicurezza di stringstream:

```
int CalcolaEtà(const string& txtAnnoDiNascita) {
    return AnnoAttuale -
                    lexical_cast<int>(txtAnnoDiNascita);
}
```

In caso di errore nel parsing, lexical_cast lancia un'eccezione di tipo **bad_cast**. In questo modo, se lexical_cast fallisce, si potrà intervenire con un opportuno blocco *try/catch*. Ma in questo caso specifico, è ancor meglio lasciare l'incombenza della gestione dell'eccezione al chiamante. In questo modo, infatti, è possibile eseguire molte validazioni, usando un unico blocco try-catch.

```
int DialogInformazioniSulGiocatore::Convalida() {
    try {
        int età = CalcolaEtà();
        int livello = CalcolaLivello();
        //... altre eventuali valutazioni ...
    } catch() {
        throw(
            "Preghiamo il Simpatico Giocatore"
            " di consultare un buon analista";
        );
    }
}
```

Poiché lexical_cast è una funzione di conversione generica, può anche funzionare in maniera inversa, trasformando, ad esempio, un intero in una stringa. Ecco *CreaStringaGameOver* implementata con lexical_cast, sfruttando il semplice collegamento fra oggetti string.

```
string Partita::CreaStringaGameOver() {
    return "Complimenti " + nome_ + "!\nHai finito il
                                        gioco in " +
     lexical_cast<string>(secondi_) + " secondi, " +
     "con un punteggio di " +
                    lexical_cast<string>(punteggio_);
}
```

Va notato che lexical_cast è implementato, internamente, usando gli stream. Quindi, tutte le considerazioni di ordine prestazionale viste per stringstream si applicano anche a lexical_cast. Inoltre non è possibile utilizzare formattatori e manipolatori con lexical_cast, e la scrittura, per frasi complesse, soffre di problemi di leggibilità ancor maggiori di stringstream, dovuti alla continua ripetizione di: "lexical_cast<tipo>". Per piccole operazioni, però, la semplicità e la flessibilità di questa soluzione sono impareggiabili.

NOTA

MEZZO PUNTO A FAVORE DI CSTRING

La funzione MFC CString::FormatMessage, appoggiandosi sull'API FormatMessage di kernel32, offre molte possibilità, fra le quali i segnaposti posizionali, alla boost::format. Rimangono, sia chiaro, i problemi di insicurezza rispetto ai tipi, con in più la totale dipendenza da compilatori e ambienti Microsoft.

LE TENTAZIONI DI SPRINTF

I programmatori C++ di derivazione C avranno probabilmente pensato ad un'implementazione di *CreaStringaGameOver* di questo tipo:

```
string Partita::CreaStringaGameOver() {
    //creazione buffer
    const int DimBuffer = 255;
    char buffer[DimBuffer];

    //riempimento
    static const char* messaggio =
      "Complimenti, %s!\n Hai finito il gioco"
      "in %.2f secondi con un punteggio di %d\n";
    sprintf(buffer, messaggio, nome_.c_str(), secondi_,
                                        punteggio_);

    //costruzione implicita std::string(char[])
    return buffer;
}
```

Nella prima fase viene allocato un buffer, la cui memoria viene riservata direttamente sullo stack. Ciò è vantaggioso sotto il profilo delle prestazioni, dal momento che il buffer viene allocato una volta sola, e resta sotto diretto dominio del programmatore. Questo sistema sarà quindi più rapido di stringstream, colpendo così il programmatore C++ medio in uno dei suoi punti deboli: il complesso da ansia di prestazioni. Proprio mentre state cercando di resistere all'attrazione di usarlo, sprintf vi assale con un'altra tentazione, ancora più subdola: il grande vantaggio di avere **il testo di formato** (*nell'esempio, messaggio*) **separato dalle variabili** (*nome_, secondi_, punteggio_*). Oltre a rendere il codice indubbiamente più pulito e leggibile, questa caratteristica permette a sprintf una versatilità che stringstream non ha. Vediamo perché. Supponiamo di voler lanciare il nostro gioco sul mercato europeo, permettendone la localizzazione in vari paesi. La classe Partita potrebbe cambiare così

```
class Partita {
    enum Lingua {
        Italiano = 0,
        English = 1,
        Français = 2,
        //... eccetera
    };

    Lingua lingua_;
   //[...] resto dell'implementazione [...]
};
```

Stampare un messaggio diverso per ogni linguaggio con sprintf è molto semplice: basta trasformare messaggio in un array.

```
static const char* messaggio[] = {
   "Complimenti, %s!\n Hai finito il gioco "
   "in %.2f secondi, con un punteggio di %d\n",

   "Congrats, %s!\n You ended the game "
   "in %.2f seconds, with a score of %d\n",

   "Bravo, %s!\n Tu has fini le jeu "
   "en %.2f seconds, avec un score de %d\n "

   //eccetera...
}

sprintf(buffer, messaggio[lingua_], nome_.c_str(),
                                  secondi_, punteggio_);
```

Ovviamente l'approccio usato qui è didattico, e in un'implementazione reale i messaggi verrebbero recuperati direttamente da un file di risorse (magari in una sottodirectory diversa a seconda del linguaggio), senza alcun bisogno di usare enum e vettori, e permettendo di effettuare qualsiasi aggiornamento senza dover ricompilare una sola riga di codice.
Si può capire quanto giochi a favore di sprintf la possibilità di effettuare operazioni di questo tipo, e per giunta in maniera banale. Provate, per contro, ad implementare qualcosa di simile usando stringstream. Già in questo semplice caso avreste bisogno o di un if chilometrico, o di memorizzare nel file tre pezzi di messaggio distinti. Da questo punto di vista gli stream ricordano molto da vicino il Commodore 64. E non è un complimento.

SPRINTF E COMPAGNI NON SONO UNA SOLUZIONE

Un buon programmatore C++ deve saper resistere alle tentazioni di sprintf, ricordandosi che si tratta di una funzione di libreria C, e che – come abbiamo visto per *atoi* -, usare direttamente le funzioni C nel C++ è quasi sempre una pessima idea. Sprintf non fa eccezione. Cominciamo a notare che sprintf lavora con il sistema dell'ellissi – un incubo, dal punto di vista della sicurezza rispetto ai tipi. Scrivendo:

```
sprintf(buffer, "%s", variabile);
```

Non possiamo sapere se l'operazione andrà a buon fine o meno, perché non conosciamo il tipo di "variabile". Se variabile fosse un char[] (ben formattato), allora funzionerebbe tutto bene. Se variabile fosse un qualsiasi altro tipo, invece, il tipo richiesto nel messaggio e quello della variabile non corrisponderebbero più. Quindi il caso seguente produrrebbe effetti catastrofici:

VERSIONI ESTESE

Tutte le soluzioni viste nell'articolo (quelle non-sconsigliate, almeno), possono essere usate per stringhe Unicode, attraverso l'uso dei caratteri estesi. boost::lexical_cast si adatta esplicitamente a qualunque tipo. wstring e wstringstream e boost::wformat sono dei typedef rapidi per specializzare queste classi per wchar_t.

```
int variabile = 17;
sprintf(buffer, "%s", variabile);
```

Noi umani possiamo accorgercene. Il compilatore, invece, non può, dal momento che i parametri passati per ellissi non vengono considerati a tempo di compilazione, ma solo dinamicamente, durante l'esecuzione del programma. Quindi: **sprintf non è sicuro rispetto ai tipi**. Va poi notato che **sprintf funziona solo con tipi primitivi,** e non con tipi definiti dall'utente. Stringstream, al contrario, funziona con qualunque tipo preveda un operatore di inserimento in stream. Inoltre possiamo chiederci cosa succederebbe, in un caso simile:

```
char buffer[25];
sprintf(buffer, "StringaDiVentiseiCaratteri");
cout << buffer; //crash!
```

Qui stiamo tentando di inserire una stringa di 26 caratteri (27, in realtà), in un buffer di 25. Il compilatore ci avvertirà dell'errore? Neanche per idea: la computazione avrà senso soltanto a tempo di esecuzione.
Cosa succederà al buffer? **Andrà in overflow**, dando origine ad un bug insidioso e ad una potenziale falla di sicurezza. Il mondo soffrirebbe molte vulnerabilità critiche in meno se i programmatori la smettessero di usare funzioni come sprintf, strcpy, e compagne. Alcuni compilatori permettono un'estensione di sprintf chiamata snprintf (o_snprintf), che quantomeno tronca il buffer in caso di overflow. La stringa risultante si interromperà di colpo, ma se non altro saremo al sicuro.

```
char buffer[25];
snprintf(buffer, 25, "StringaDiVentiseiCaratteri");
cout << buffer; //crash?
```

Purtroppo, **snprintf non è una funzione C++ standard**, e il codice che dovesse usarlo smetterebbe di essere portabile. Inoltre il comportamento di snprintf può variare a seconda delle implementazioni. Buffer potrebbe contenere il valore: "StringaDiVentiseiCaratte\0", con un terminatore aggiunto dalla funzione alla fine. Oppure potrebbe non farlo, e inserire nel buffer il valore: "StringaDiVentiseiCaratter" – cosa che provocherebbe un disastro al primo utilizzo. Nell'esempio, verrebbe stampato qualcosa come "StringaDiVentisei Caratter[@#@24", seguito da un'altra marea di spazzatura e da un bel crash per segmentation fault.
Il mio consiglio personale è di **non usare mai sprintf**, e preferire qualsiasi alternativa C++, usando **snprintf** solo in quell'un percento dei casi in cui i vincoli prestazionali non permettono altro. E sempre **dopo** *aver realmente tentato* di risolvere il problema con funzionalità più sicure.

UNA CSTRING NON È UNA SOLUZIONE

Va bene il discorso sulle prestazioni, ma quello sulla flessibilità di avere il testo separato dalle variabili, con l'esempio sull'applicazione multilingue? Quello era interessante. Non possiamo inventarci una "via C++" per risolvere il problema? Certo! A patto di chiarire cosa sia una via C++. Cominciamo dalla *pars destruens*, spiegando cosa, sicuramente, non lo è. Scrivere una classe String tutta nostra, nella quale mettere una funzione "Format" che si comporti come sprintf non è un'idea molto valida. Siccome questa è una rubrica votata al cross-platform (e anche un po' alla par-condicio), prenderemo come esempi negativi due classi derivanti da framework di mondi opposti: wxString (da wxWidgets – open source) e CString (da MFC - proprietario). Partiamo da un presupposto: scrivere una classe String quando ce n'è già una standard e ben collaudata, ha poco senso. Qui i due framework non hanno colpa, dato che entrambi sono nati *molto prima* che la libreria standard diventasse *abbastanza standard* da costituire un punto di riferimento. Oggi, però, non ha più senso ricorrere a simili non-soluzioni, e le classi già esistenti dovrebbero essere segnalate come *deprecated* (punto a favore di wxWidgets: wxString non sarà più supportata in futuro). Anche il design di queste classi non è esaltante. Ammassare un mucchio di funzioni membro in una classe anabolizzata, non significa scrivere in C++. Si consideri come controesempio l'implementazione della libreria STL: una piccola classe string che definisce costruttori, distruttore, operatori e conversioni; e una grande libreria di algoritmi scritti come funzioni-non-membro. Queste sono considerazioni generali. Andando nel particolare, le due funzioni Format esposte da wxWidgets e CString sono semplicemente due wrapper costruiti attorno di snprintf (quando va bene. Per wxWidgets, a seconda del compilatore, si può ricadere su sprintf. Brr…). Morale del paragrafo: se bisogna usare una funzione insicura, tanto vale usarne una standard!

LA SOLUZIONE BOOST::FORMAT

Va bene, va bene: scrivere una classe onnicomprensiva con una funzione Format che faccia da wrapper a sprintf non è una soluzione. Qual è, allora? Presto detto: scrivere una classe specifica, che permetta la formattazione della stringa, in modo sicuro rispetto ai tipi e agli overflow. Fortunatamente, non c'è biso-

gno di reinventare la ruota. Samuel Krempp ha fatto un ottimo lavoro con **boost::format**, che è una soluzione efficace, cross-platform e riveduta da una grande comunità come boost. È possibile accedere a format includendo l'header <boost/format.hpp>. Possiamo vedere un format come un oggetto simile a stringstream (internamente, infatti, è implementato per mezzo di stringstream), che, però, permette di associare un testo di formato, nel costruttore. Una volta ottenuto un oggetto format, è possibile inviarlo in uno stream, o convertirlo a stringa, con la funzione str().

```
format saluto("Ciao, mondo!");
cout << saluto;
string str = saluto.str();
```

Certo, visto così non è granché. Ma il bello è che il testo di formato può contenere degli specificatori, proprio come sprintf. Gli argomenti non verranno passati per ellissi, ma tramite l'operatore %, in un secondo tempo. Ecco, quindi, una nuova versione di *CreaStringaGameOver*:

```
string Partita::CreaStringaGameOver() {
    format messaggio (
        "Complimenti, %s!\nHai finito il gioco "
        "in %.2f secondi, con un punteggio di %d\n"
    );
    messaggio % nome_ % secondi_ % punteggio_;

    return messaggio.str();
}
```

ROBERTO ALLEGRA
Il sito www.robertoallegra.it contiene l'elenco degli articoli pubblicati in questa rubrica, con eventuali approfondimenti e errata corrige, e la possibilità di contattare l'autore per richieste, critiche e suggerimenti.

Ha senz'altro l'eleganza, la concisione e la versatilità di sprintf. Ma non ne porterà anche i difetti? Certamente non c'è il rischio di buffer overflow, dal momento che non c'è alcun buffer visibile! Qualcuno potrebbe però chiedersi che succederebbe se il tipo passato non corrispondesse a quello indicato nel testo di formato. La risposta è: *assolutamente niente*. Gli specificatori, infatti, vengono messi soltanto per indicare possibili criteri di formattazione, ma non hanno alcun valore per l'identificazione di tipo. Pertanto: **boost::format è sicuro rispetto ai tipi e agli overflow**. A riprova di ciò, se non vi servono formattazioni particolari, potete anche evitare di specificare i tipi:

```
    format messaggio (
        "Complimenti, %1%!\nHai finito il gioco "
        "in %2% secondi, con un punteggio di %3%\n"
    ) % nome_ % secondi_ % punteggio;
```

Questa soluzione è identica alla prima, a parte il fatto che abbiamo perso lo specificatore di precisione per il secondo parametro. Abbiamo però illustrato due funzionalità importanti: si possono inserire i dati subito dopo la costruzione dell'oggetto, ottenendo così la compattezza di sprintf. Inoltre, si possono utilizzare **specificatori posizionali**. Lo specificatore %1%, infatti, identifica "il primo dei parametri passati". Ciò è fatto apposta per permettere l'internazionalizzazione del nostro gioco. Non tutte le lingue, infatti, dispongono le parole nello stesso modo all'interno della frase. Nuovo esempio: nel nostro gioco ci sono delle armi che possono appartenere a diversi eroi. Vogliamo descriverle in modo generico:

```
    const char* eroe[] = {"Re Artù", "King Arthur"};
    const char* arma[] = {"Spada", "Sword"};
    const char* msg[]  = {"%1 di 2", "%2's %1"};
    cout << format(msg[lingua]) % arma[lingua] %
                                        eroe[lingua];
```

In italiano verrà stampato "Spada di Re Artù", mentre la versione inglese invertirà le variabili: "King Arthur's Sword". Senza gli specificatori posizionali, avremmo dovuto evitare il genitivo sassone con perifrasi innaturali, o includere il cambiamento nel codice, sotto forma di interminabili *if*. Boost::format prevede tante altre funzionalità (così tante, che qualcuno critica questa soluzione perché "troppo general-purpose" e quindi "troppo pesante"), e tanti altri vantaggi. Dall'uso delle eccezioni ad una sintassi avanzata degli specificatori di formato. Spero di avervi incuriosito abbastanza da consultarne la documentazione!

CONCLUSIONI

Con un solo passo nel mondo del text processing abbiamo già fatto un bel viaggio intorno alle soluzioni che permettono di creare una stringa, o estrarne dei dati, analizzandone le caratteristiche e le debolezze, facendo anche riferimento all'internazionalizzazione del software. Se vi siete persi in mezzo a tante possibilità, permettetemi di tirare le somme: per semplici conversioni da e verso stringhe, usate boost::lexical_cast. Se tali operazioni dovessero complicarsi, potete scegliere: se sono molto complicate o volete supportare l'internazionalizzazione, usate boost::format; se non avete grosse pretese, o la conversione avviene da stringa, usate stringstream. Se, in un caso molto specifico, dopo aver provato tutte queste soluzioni, avete bisogno di prestazioni maggiori… armatevi di pazienza e debugger, usate *snprintf* e fate molta, molta attenzione.

Roberto Allegra

MIGRARE FACILMENTE A MS SQL SERVER

AVETE SVILUPPATO UN'APPLICAZIONE CHE SI FONDA UN DATABASE ACCESS O ORACLE E VI VIENE CHIESTO DI MIGRARE A MICROSFT SQL SERVER? NIENTE PANICO, ECCO GLI STRUMENTI RAPIDI PER MIGRARE DATI E STRUTTURE SENZA PROBLEMI

Dopo aver sviluppato la nostra incredibile piattaforma di gestione dell'intera produzione aziendale, improvvisamente il nostro capo ci informa che dobbiamo migrare la nostra architettura da Oracle a Microsoft SQL Server. Se è vero che da un punto di vista dell'applicazione siamo stati previdenti e ci basterà cambiare la stringa di connessione per migrare l'intera applicazione, diverso è il discorso della migrazione della struttura e dei dati aderenti al DB.

Affrontiamo le problematiche che ci troveremo di fronte in una migrazione. Nella migrazione si deve tener conto dello schema e dei dati, ciò vuol dire trasformare alcuni elementi essenziali quali la struttura delle Tabelle (compresi indici e Constraints) ed i record in esse contenuti. Inoltre per database più evoluti, si devono trasferire anche le Stored procedures, i triggers ed altri oggetti ugualmente importanti.

Si potrebbe pensare ad una migrazione manuale, ma questo comporterebbe certamente alcuni mesi di lavoro, un costo eccessivo, ed un gran numero di test per verificare che l'applicazione convertita, alla fine del processo di migrazione funzioni correttamente.

Per evitare questi rischi, ci viene in aiuto *SQL Server Migration Assistant (SSMA). SSMA* è una suite completa di strumenti di migrazione, prodotta da Microsoft dal cui sito è possibile effettuare il download gratuito. La suite mette a disposizione gli strumenti idonei per la valutazione.

Gli obiettivi dichiarati sono:

- Ridurre i rischi di conversione/migrazione
- Garantire una migrazione corretta
- Garantire che i database migrati siano realmente funzionanti con le applicazioni progettate su di esse.
- Assicurare che il nuovo database funzioni allo stesso modo del DB originale
- Ridurre tempi e costi di migrazione
- Automatizzare parzialmente o completamente, ampie porzioni di un progetto di conversione con SSMA

SSMA PER ACCESS

Dopo aver installato SSMA for Access, troveremo la nostra bella iconcina sul desktop, lanciato il programma si apre la finestra di dialogo: License Management in cui ci viene chiesto di scaricare la licenza dal sito Microsoft. Cliccando sul link license registration page veniamo portati direttamente nella pagina corretta in cui dobbiamo registrarci su Windows Live Id, se avete già un account Hotmail, Msn o Microsoft Passport potete utilizzarlo senza registrarvi nuovamente.

Fig. 1: Per usare SSMA è necessario ottenere una licenza. Per farlo è necessario registrarsi

Dopo aver completato la registrazione dobbiamo cliccare su Finish in modo da poter effettuare il download del file (*access-ssma.license*) con la licenza, sull'hard disk. A questo punto possiamo copiare il file nella cartella di installazione di *SSMA* (il percorso di default è c:\programmi\Microsoft Sql Server Migration Assistant). Ritornati alla finestra di dialogo: *License Management*, dobbiamo cliccare sul pulsante *Refresh Licens*

DOVE TROVO SSMA?

SQL Server Migration Assistant, si può scaricare gratuitamente da: http://www.microsoft.com/sql/solutions/migration/default.mspx

***Fig. 2:** Dopo aver cliccato su refresh la licenza è installata*

Siamo pronti per iniziare, appena lanciamo il programma, si apre un Wizard che ci aiuta ed eseguire i passi corretti:

1 Nella finestra di dialogo *Create New Project*, dobbiamo scrivere il nome del nuovo progetto di migrazione, e selezionare il percorso di salvataggio

2 Si apre la finestra di dialogo *Add Access database*. In questa finestra dobbiamo inserire il percorso completo del database da convertire. Cliccando su *Add* si apre la classica finestra di dialogo in cui possiamo navigare nelle nostre directory per selezionare il database desiderato. Cliccando su *Find* si apre la finestra di dialogo *Find Database Wizard*

3 Nella finestra di dialogo *Find Database Wizard* possiamo cercare i database Access per un ampio numero di filtri: in base al nome, alla data di creazione, alla dimensione, ecc. Dopo aver selezionato il percorso in cui cercare il database possiamo cliccare su *Add* ed avviare la ricerca premendo *Scan*

4 Nella finestra di dialogo *Select File* verranno elencati tutti i database che corrispondono ad i criteri di ricerca. A questo punto possiamo selezionare i database da inserire nel progetto e cliccare su *Next*. Ci viene richiesto di fare una ulteriore verifica del database selezionato e cliccare su *Finish* per ritornare alla finestra precedente. A questo punto, si deve cliccare su *Next* per andare avanti

5 Si apre la finestra di dialogo *Select Object to migrate*. In questa finestra possiamo scegliere gli oggetti da migrare, quindi si possono selezionate le tabelle o le query interessate al processo di migrazione.

6 Nella finestra di dialogo *Connect to Sql Server 2005*, possiamo impostare i nostri parametri per la connessione ad *Sql Server 2005*. Se nel nome del database, indichiamo un database inesistente, allora ne viene creato uno nuovo.

7 A questo punto viene avviato il processo di migrazione, che prevede tre passi: conversione degli oggetti selezionati dal database *Access* di origine, caricamento degli oggetti convertiti nel database di destinazione *Sql Server 2005*, migrazione dei dati contenuti negli oggetti selezionati.

8 Al termine del processo di migrazione è possibile visualizzare dei report dettagliati sull'esito del processo. Sono presenti tutte le informazioni inerenti la migrazione sia in toto che in dettaglio sul singolo oggetto. Le operazioni per migrare il classico database Access *NWIND* sono state completate in alcuni secondi, il report finale, invece, prevede un tempo pari ad un ora nel caso di un processo di migrazione manuale.

L'AMBIENTE DI SSMA FOR ACCESS

L'ambiente di *SSMA for Access*, è particolarmente curato e presenta i classici elementi di una applicazione Windows: Menu, Toolbar, finestre di esplorazione e pannelli di controllo. Se non vogliamo utilizzare il wizard, per avviare la migrazione dobbiamo:

- Creare un nuovo progetto selezionando la voce di menu: *File/New Project*, oppure dal pulsante corrispondente sulla toolbar
- Selezionare i database Access da migrare ed aggiungerli nel progetto selezionando la voce di menu: *File/Add Databases*, oppure dal pulsante corrispondente sulla toolbar
- Selezionare gli oggetti da migrare nel pannello *Access Metadata Explorer*
- Connetterci ad un'istanza di Sql Server 2005 selezionando la voce di menu: *File/Connect to Sql Server*

Analizziamo in dettaglio gli elementi dell'interfaccia disponibili: (Figura 3)

Fig. 3: Gli elementi dell'interfaccia

Metadata Explorers
SSMA contiene due pannelli metadata explorers che permettono di navigare e compiere azioni sui database *Access* coinvolti nel processo di migrazione, e sul database *sql server* di destinazione

Access Metadata Explorer
Il pannello *Access Metadata Explorer* mostra informazioni circa i database *Microsoft Access* che abbiamo aggiunto al progetto. Quando aggiungiamo un database *Access*, *SSMA* analizza e visualizza automaticamente tutte le informazioni inerenti il database.
Nel pannello Access Metadata Explorer, possiamo compiere le seguenti operazioni:

- Navigare nelle tabelle di ogni database *Access*.
- Selezionare gli oggetti da convertire nell'equi-

VERSIONI DI SSMA

SQL Server Migration Assistant è attualmente disponibile in tre versioni per migrare dai seguenti database

- **Access**
- **Oracle**
- **Sybase**

Sono in fase di sviluppo ulteriori due versioni per migrare dai database Db2 e Informix

FACILITIES

L'integrazione con Visual Studio 2005 prevede un ulteriore caratteristica molto utile come l'evidenziazione automatica per la sintassi e l' IntelliSense in Transact-SQL.

valente sintassi Transact-SQL

- Selezionare gli oggetti coinvolti nella migrazione dei dati. La migrazione dei dati è di norma una operazione di bulk-copy in cui vengono caricate le righe contenute in ogni tabella. Il numero di righe da caricare in ogni transazione Sql Server può essere configurato nel project settings
- Collegare e modificare i collegamenti dagli oggetti *Access* di partenza alle tabelle *SQL Server 2005* di destinazione.

SQL Server Metadata Explorer

Il pannello *SQL Server Metadata Explorer* visualizza le informazioni inerenti l'istanza di SQL Server 2005 selezionata. Quando ci connettiamo ad un'istanza di SQL Server 2005, SSMA analizza automaticamente le informazioni sui metadata dell'istanza, e le memorizza nel file di progetto. Possiamo utilizzare SQL Server Metadata Explorer per selezionare gli oggetti convertiti di Access, e sincronizzarli con i corrispondenti oggetti nell'istanza di SQL Server 2005.

Metadata

A destra di ogni pannello *metadata explorer* sono presenti dei pannelli che descrivono gli oggetti selezionati. Per esempio, selezionando una tabella nel pannello *Access Metadata Explorer*, nel pannello corrispondente sono visualizzate: la Tabella, Type Mapping, Proprietà, e Dati. Selezionando una tabella in *SQL Server Metadata Explorer*, nel pannello corrispondente sono visualizzate: la tabella, le proprietà SQL, i Dati. Molti metadata visualizzati, sono a sola lettura. Tuttavia, è possibile modificare i seguenti metadata:

- In *Access Metadata Explorer*, possiamo modificare i *type mappings*. Queste modifiche si dovrebbero fare prima di avviare il processo di conversione degli schemi e creazione dei reports.
- In *SQL Server Metadata Explorer*, possiamo modificare le tabelle e gli indici delle tabelle. Queste modifiche si dovrebbero fare prima di avviare il processo di caricamento degli schemi in Sql Server

The Project Toolbar

La project toolbar contiene i pulsanti per lavorare con i progetti: Add database, connecting to SQL Server. I classici pulsanti di Nuovo, Salva e cancella presenti nel menu File, ed il pulsante che avvia il wizard analizzato in precedenza.

Migration Toolbar

La migration toolbar contiene i seguenti comandi:

- *Convert, Load, and Migrate*: E' il comando che esegue, tutti in una volta, i tre passi del processo di migrazione: conversione degli oggetti selezionati dal database *Access* di origine, caricamento degli oggetti convertiti nel database di destinazione *Sql Server 2005*, migrazione dei dati contenuti negli oggetti selezionati.
- *Create Report*: Converte lo schema del database Access selezionato, nella corrispondente sintassi SQL Server, e crea i report che mostrano i dati con gli oggetti che si riescono a trasformare. Questo comando è disponibile soltanto quando gli oggetti sono selezionati in Access Metadata Explorer.
- *Convert Schema*: Converte lo schema del database Access selezionato, nel corrispondente schema SQL Server. Questo comando è disponibile soltanto quando gli oggetti sono selezionati in Access Metadata Explorer.
- *Load to Database*: Carica lo schema selezionato in SQL Server 2005. Se gli oggetti selezionati non esistono, allora, vengono creati in SQL Server, Altrimenti vengono modificati gli oggetti esistenti. Questo comando è disponibile soltanto quando gli oggetti sono selezionati in SQL Server Metadata Explorer.
- *Migrate Data*: Permette di migrare i dati dal database Access a SQL Server 2005. Questo è l'ultimo passo di un processo di migrazione, pertanto, prima di lanciare questo comando, si devono lanciare i comandi *Convert Schema e Load to Database*. Questo comando è disponibile soltanto quando gli oggetti sono selezionati in Access Metadata Explorer.
- *Stop*: Termina il processo di migrazione corrente.

I pannelli Output ed Error List

Il pannello di *Output* mostra i messaggi di stato di SSMA durante la conversione degli oggetti, la sincronizzazione degli oggetti e la migrazione dei dati.

Il pannello *Error List* mostra gli errori, i warning, ed i messaggi informativi in una lista che è anche possibile ordinare

MIGRARE DA ORACLE

Diamo ora uno sguardo, ai concetti che si devono affrontare per migrare ad *Sql Server 2005* da *Oracle*. Dopo aver installato Microsoft SQL Server Migration Assistant si deve installare L'Extension Pack. SSMA è il tool che permette di convertire il dialetto SQL utilizzato da *Oracle* (*PL/SQL*) nel dialetto SQL utilizzato da Sql Server 2005 (Transact-SQL), mentre L'*Extension Pack* si occupa di creare un data-

base *sysdb* in *Sql Server* e installa le procedure necessarie per emulare le funzioni di *Oracle*.
Per installare SSMA è sufficiente lanciare il programma di Setup. L'unico prerequisito è la presenza della java runtime 1.4.2 che si può eventualmente scaricare da:

http://java.sun.com/j2se/1.4.2/download.html

Anche per questa versione di *SSMA*, appena si lancia il tool, ci viene richiesto di scaricare la licenza dal sito *Microsoft*. (**Figura 4**) Le modalità sono le stesse viste in precedenza, l'unica differenza è che dal sito ci viene richiesto di cliccare sul link *ssma-license key*.
L'Extension Pack installa due database di test

Fig. 4: È necessario installare una licenza anche per Oracle

test_platform in Oracle e *test_platform_db* in MS SQL Server, la condizione necessaria, affinché venga installato il database *test_platform*, è la presenza di un client Oracle connesso ad un server Oracle. Nella migrazione da un database *Oracle ad Sql Server*, dobbiamo tenere conto di alcune differenze tra i due database:

- Diverso modello di gestione delle transazioni
- Data types diversi
- Gestione eccezioni (solo SQL Server 2000)
- Packages e package objects
- Sequenze (Oracle) vs. identities (MSSQL)

Per gestire queste differenze possiamo utilizzare le opzioni che il tool ci mette a disposizione, in particolare possiamo selezionare le seguenti voci:

- Generate ROWID column: Permette di generare nello schema di uscita una colonna rowid. Questa opzione riduce notevolmente la percentuale di conversione di Trigger e Sp
- Allow sequence-to-identity conversion: Trasforma sequence in colonne identity. Questa è la strada da preferire laddove sia possibile
- Allow insertion to IDENTITY columns: Una volta selezionato, permette l'inserimento in colonne di tipo identity. In questo caso si deve verificare la compatibilità con la logica business
- Show system sequences: Mostra le finestre di dialogo per accedere alle sequenze di sistema. é possibile attivare questa opzione soltanto se "Insert into identity" è attivo
- Convert transaction processing statements: Abilita o disabilita la conversione di comandi SQL per le transazioni
- Convert exceptions_ Abilita o disabilita la conversione per la gestione di eccezioni

Possiamo facilmente comprendere che ci troviamo di fronte a problematiche più complesse rispetto alla semplice migrazione di un database *Access*. La suite mette a disposizione un potente strumento di valutazione della migrazione. Questo strumento ci permette di valutare la complessità ed il tempo necessario per la migrazione ed inoltre ci fornisce una stima, in percentuale, degli oggetti immediatamente convertibili, in particolare, permette di valutare:

- Numero totale di righe di codice.
- Numero totale di statement *select, insert, delete e update*
- Numero totale di *cursori, record, e eccezioni.*
- Numero totale e percentuale di componenti convertibili automaticamente.
- Stima della complessità di migrazione.
- Stima dei tempi necessari per migrare manualmente.

Il processo di migrazione attraversa le seguenti fasi:

- Valutazione del progetto: Con lo strumento descritto in precedenza
- Migrazione dello schema: A questa punto vengono generati gli script di creazione degli oggetti dello schema, in questa fase avviene la conversione da PL/SQL a T-SQL con particolare attenzione alle differenze di:Tipi, operatori, oggetti e sintassi
- Migrazione dei dati: I due server vengono configurati come linked server e le colonne sono mappate automaticamente, dopo la conversione viene creato un report di migrazione
- Migrazione logica di business: Viene convertito il codice contenente logica di business ed avviene la conversione di funzioni e trigger
- Test codice migrato: In questa fase è possibile testare viste, funzioni, stored procedure ed sql statements

In ogni caso SSMA rappresenta un ottimo strumento per favorire la migrazione dei dati da un ambiente qualunque a Microsoft SQL Server

Luigi Buono

COSA SI PUÒ MIGRARE?

Gli oggetti coinvolti nel processo di migrazione sono:
SQL statements
Stored procedures
Triggers
Viste
Record
Cursori
Sequenze
User-defined functions
Packaged functions
Tabelle
Indici
Constraints
Defaults

SOFTWARE SUL CD

EZPublish 3.9.0

IL FRAMEWORK PER LA COSTRUZIONE DI CMS

Sebbene ad un primo sguardo, dopo l'installazione, potreste pensare di trovarvi davanti ad un normale sistema CMS, EXPublish è molto di più. Si tratta infatti di un completo framework RAD per la costruzione di applicazioni di gestione di contenuti molto specializzato. E' possibile definire nuove classi, nuovi formati per l'input, definire il workflow dell'applicazione, elaborare template molto accurati utilizzando smarty. Si tratta insomma di un prodotto molto evoluto con potenzialità straordinarie

Directory:/Ezpublish

APACHE 2.3.4

IL WEB SERVER PIÙ USATO AL MONDO

L'ultima versione del Web Server progettato da Apache e che da solo tiene in piedi una buona parte di Internet. Nonostante l'agguerrita concorrenza Apache rimane un Web Server incredibilmente usato. I suoi moduli sono praticamente illimitati, è incredibilmente leggero,viene utilizzato praticamente su tutte le piattaforme di hosting al mondo. Se avete in mente di progettare un'applicazione Web non potete fare a meno di avere installato sulla vostra macchina in locale una versione di Apache. Quella che vi presentiamo è la versione 2.3 della serie 2, che ormai è sufficientemente consolidata da essere usata in ambienti di produzione

Directory:/Apache

DRUPAL 5.1

IL COSTRUTTORE DI BLOG

Drupal è prima di tutto un sistema CMS, consente cioè la pubblicazione dei contenuti in modo semplificato, di modo che anche gli utenti meno smaliziati possano inserire dei contenuti su Internet pur non avendo conoscenze di programmazione o di html, ma Drupal è anche molto di più. Utilizzando Drupal ognuno degli utenti iscritti al sito diviene proprietario di un blog e con pochi accorgimenti potrete fare in modo che ciascuno di questi blog goda di una vita propria indipendente dal sito principale, oppure se lo desiderate potete far comunicare i vari blog in modo da creare una vera community di bloggers

Directory:/Drupal

MEDIAWIKI 1.9.2

LA VOSTRA WIKIPEDIA PERSONALIZZATA

Un wiki è un sito internet gestito da qualcosa molto simile ad un CMS. Così come un CMS viene utilizzato per l'inserimento di rapido di contenuti sul Web, a differenza di un CMS però dispone di una sorta di linguaggio interno tale che se nel testo vengono inseriti dei marcatori particolari, automaticamente viene creata una pagina che li referenzia. Questo sistema garantisce la creazione rapida di enciclopedie che sfruttano al massimo il sistema di HyperLink tipico di internet. Inoltre le pagine vuote che vengono create automaticamente dal sistema possono essere riempite dagli utenti che ne hanno i permessi e così via con un meccanismo piramidale. Comprenderete che creare un'enciclopedia utilizzando questo sistema è piuttosto semplice. MediaWiki è il software che serve una delle più grandi enciclopedie mondiali online: wikipedia

Directory:/Mediawiki

HIBERNATE 3.2.2

IL RE DEI ORM

Nessun linguaggio sfugge alla logica secondo cui i database relazionali sono difficilmente rappresentabili come oggetti. Ed ecco che arriva Hibernate che

garantisce il mapping fra oggetti ed elementi di un database relazionale superando i limiti imposti dalle due diverse tecnologie. Il tool è particolarmente interessante, sia per l'elevato grado di disaccoppiamento che offre tra gli elementi del database e le classi che lo gestiscono, sia per la capacità di trasformare dati tipicamente trattati in modo relazionale in classi ed oggetti

Directory:/Hibernate

IBATIS 1.6.0

L'ORM SEMPLICE

Ormai lo sappiamo:linguaggio SQL e programmazione ad oggetti non vanno d'accordo. Per superare le evidenti incongruenze di questa improbabile accoppiata sono nati gli ORM, Object Relational Mapping, tools in grado di eseguire una mappatura di elementi SQL in corrispondenti oggetti e classi. Il capostipite di questa progenie di tool è senza dubbio Hibernate, ma se non volete cozzare contro la complessità di questo enorme strumento, allora iBatis è quello che fa per voi. E' leggero, semplice da usare e dispone di tutte le funzionalità di un ORM professionale. Usatelo se SQL comincia ad andarvi stretto

Directory:/iBatis

MYSQL 5.1.15

IL PRINCIPE DEI DATABASE

Indispensabile per programmare webapplication in tecnologia PHP. Non che non sia possibile utilizzare altri database, ma MySQL e PHP rappresentano veramente un binomio inscindibile. L'integrazione fra questo database e il linguaggio di scripting più usato sulla rete è talmente alta da fare divenire quasi un obbligo l'uso congiunto di questi due strumenti

Directory:/Mysql

POSTGRES 8.2.3

IL GRATIS COMPLETO E VELOCE

Nessun server di database offre la completezza delle funzioni esposte da PostgreSQL e rimane comunque completamente Gratis. PostgreSQL è probabilmente il più estendibile fra i database esistenti. Inutile parlare della gamma praticamente completa delle sue funzioni. Il punto di forza che lo pone probabilmente al di sopra di tutti i concorrenti rimane l'alta possibilità di personalizzazione, oltre, naturalmente, alla velocità, alla stabilità ed al costo nullo. Unica pecca, una certa complessità nella gestione. E' sicuramente da usare in ambienti di produzione professionali che non possono accontentarsi di alcun compromesso

Directory:/Postgres

FIREBIRD 2.0.0.12748-0

IL DATABASE VELOCE COME UN FULMINE

Ha attraversato una serie incredibile di disavventure. Acquistato da Borland è poi tornato ad essere OpenSource. Presente sul mercato da tempo immemorabile è riuscito a sopravvivere alle sue varie vicissitudine solo grazie alle sue grandi doti tecniche. E' un database velocissimo e ultraleggero, dotato però di tutte le funzioni di un server professionale

Directory:/Firebird

DADABIK 4.2

UN CREATORE DI INTERFACCE VERSO DATABASE

Si tratta di una web application scritta in PHP che consente di costruire altre web application basate su database. Ad esempio se volete costruire un sito che esponga un catalogo multimediale, non vi resta che informare Dadabik di quali campi si compone il catalogo in questione, di quali funzionalità volete che il vostro sito sia dotato, e lasciare a DadaBik il compito di generare la vostra interfaccia.Non è necessario avere competenze estese di programmazione, l'uso di DadaBik è piuttosto semplice.

Directory:/Dadabik

IRRLICHT 1.2

ACCENDI IL MIGLIORE MOTORE 3D OPEN SOURCE!

Irrlicht è un motore per la grafica tridimensionale, scritto in C++ e utilizzabile sia con questo linguaggio, sia con la tecnologia .NET.Presenta le principali caratteristiche di un motore professionale e vanta una notevole comunità di sviluppatori, con diversi progetti in attivo. Irrlicht ha tra i suoi pregi anche quello di potere utilizzare sia DirectX che OpenGL per cui diventa particolarmente adatto anche per lo sviluppo di giochi multipiattaforma

Directory:/Irrlicht

JAVA SE DEVELOPMENT KIT 6

IL COMPILATORE INDISPENSABILE PER PROGRAMMARE IN JAVA

Se avete intenzione di iniziare a programmare in Java oppure siete già dei programmatori esperti avete bisogno sicuramente del compilatore e delle librerie Java indispensabili. Sotto il nome di Java SE Development Kit vanno appunto tutti gli strumenti e le librerie nonché le utility necessarie per programmare in JAVA. L'attuale versione è la 6.0, ovvero la nuovissima release densa di innovazioni e molto più legata al desktop di quanto non fossero tutte le precedenti

Directory:/J2SE

ECLIPSE SDK 3.2.2

L'IDE TUTTOFARE

Eclipse è un progetto completo portato avanti da Eclipse Foundation con la collaborazione di una miriade di aziende fra cui IBM, Adobe, Sun e che si è prefissata lo scopo di creare un IDE estendibile per plugin adattabile a qualunque tipo di linguaggio o tecnologia. Di default Eclipse si

propone come IDE per Java ed è qui che da il meglio di se. Ma proprio grazie ai suoi plugin è possibile utilizzarlo come ambiente di programmazione per PHP, per C++, per Flex e per molti altri linguaggi ancora. Inoltre sempre grazie per ciascun linguaggio sono disponibili altri plugin ad esempio per rendere l'ambiente RAD o per favorire lo sviluppo dei Web Services o altro. Insomma lo scopo è stato raggiunto completamente. Eclipse è realmente un IDE tuttofare, ormai maturo, e che serve una miriade di programmatori grazie alle sue caratteristiche di affidabilità e flessibilità. Unica nota negativa: una certa pesantezza che lo rende idoneo ad essere usato solo su PC con una dotazione hardware minima di tutto rispetto

Directory:/Eclipse

PHP 5.2.1

IL LINGUAGGIO DI SCRIPTING PIÙ AMATO DEL WEB

Sono tre le colonne portanti di Internet: PHP, APACHE e MySQL. Certo la concorrenza è forte. Asp.NET e SQL Server avanzano con celerità, ma a tutt'oggi non si può affermare che i siti sviluppati in PHP non costituiscano la stragrande maggioranza di Internet. Quali sono le ragioni del

successo di cotanto linguaggio? Prima di tutto la completezza. PHP ha di base tutto quello che serve ad un buon programmatore, raramente è necessario ricorrere a librerie esterne, e quando è proprio indispensabile farlo esistono comunque una serie di repository che rendono tutto immediatamente disponibile ed in forma gratuita. Il secondo punto di forza del linguaggio sta nella sua capacità di poter essere utilizzato sia in modo procedurale che nella sua forma ad oggetti certamente più potente e completa. Esiste un terzo punto di forza essenziale che è quello riguardante la curva di apprendimento. PHP è in assoluto uno dei linguaggi con la curva di apprendimento più bassa nel panorama degli strumenti di programmazione. Si tratta perciò di uno strumento indispensabile per chi si avvicina alla programmazione web, a meno che non intendiate scegliere strade diverse quali possono essere ASP.NET o JSP

Directory:/PHP

DEV C++ 4.9.9.2

UN EDITOR C++ A BASSO COSTO

Dev C++ è un editor distibuito su licenza GPL, come tale non ha costi relativi al diritto d'autore. Come tutto o quasi il software GPL la sua economicità non è affatto sinonimo di scarsa qualità. Al contrario DEV C++ è uno degli editor C++ più amati ed utilizzati da chi sviluppa in C++. Le caratteristiche sono notevoli. Si va dal debugger integrato, al project management, al class browser, al code completion. L'insieme di queste caratteristiche unite all'eccezionale leggerezza dell'ambiente lo rende particolarmente comodo da utilizzare per sviluppare progetti C++ anche di grandi dimensioni

Directory:/devcpp

AXIS 1.4

IL FRAMEWORK PER LA CREAZIONE DI WEB SERVICES

I Web Services costituiscono una delle tecnologie più interessanti degli ultimi anni. Nonostante che il mercato non abbia recepito commercialmente questa tecnica nel modo aspettato, sono diventati comunque un paletto importante all'interno di ogni struttura programmativa di un certo rilievo. Come realizzare un Web Services? Come farlo con Java? Questo framework mette a disposizione una serie di strumenti che rendono la complessità della programmazione di un WS estremamente bassa rispetto alla tecnologia. Si tratta di un framework importante da tenere nella cassetta degli attrezzi quando si intende sviluppare una qualsiasi architettura distribuita

Directory:/Axis

TOMCAT 6.0.9

IL SERVLET CONTAINER PER JAVA E JSP

L'idea è molto semplice. Sviluppare im Java pagine Web. Ad un primo sguardo, Tomcat potrebbe sembrare un normale Web Server. Ed in effetti è un normale Web Server! In grado di soddisfare le richieste per qualunque pagina HTML. In realtà però Tomcat è anche qualcosa in più, ovvero dispone della capacità di soddisfare le richieste per applicazioni Java. Potrebbe sembrare complesso, in realtà lo è meno di quanto sembri. Immaginate Tomcat come un grande contenitore al cui interno ci sono altri contenitori ciascuno dei quali rappresenta un'applicazione Java, che una volta richiamata costruisce una pagina Web interpretabile da un browser. Questo consente di sviluppare pagine Web utilizzando tutta la potenza della normale gerarchia delle classi Java e la sintassi e il linguaggio che qualunque programmatore Java conosce bene.

Directory:/Tomcat

J2ME POLISH 2.0

IL COSTRUTTORE DI GUI PER DEVICE MOBILI

J2ME polish è forse un precursore dei tempi. Si tratta di un software il cui scopo è supportare il programmatore nella creazione di interfacce destinate a dispositivi portatili quali ad esempio cellulari o palmari. Ovviamente la base su cui si fonda è J2ME ormai onnipresente in qualsiasi applicazione per dispositivi del genere, tuttavia invece la definizione dell'interfaccia basa le sue caratteristiche su semplici file HTML. Interessante è il fatto che J2MEpolish sia distribuito sotto licenza GPL

Directory:/J2MEpolish

JEDIT 4.3

L'EDITOR LEGGERO PER PROGRAMMATORI JAVA

Realizzato completamente in Java, jEdit è un completo editor di testi per programmatori disponibile per numerose piattaforme tra cui MacOS X, OS/2, GNU/Linux (Unix) e Windows. Il programma, per l'immediatezza e la semplicità d'utilizzo, è molto usato per scopi didattici e da programmatori non professionisti. Jedit integra un completo linguaggio di macro e dispone di un'architettura facilmente estensibile mediante l'utilizzo di numerosi plugin facilmente gestibili tramite il "plugin manager".

Inoltre nonostante sia stato creato per sviluppare prevalentemente applicazioni Java, dispone di funzionalità per indentare ed evidenziare il codice per oltre ottanta linguaggi di programmazione.

Directory:/Jedit

RUBY

IL NUOVO CHE AVANZA

Ruby, probabilmente lo conoscete tutti. E' un linguaggio nato ormai da una decina d'anni inizialmente con la pretesa di essere uno strumento matematico dalle caratteristiche avanzate. Nel tempo si è evoluto e migliorato fino a raggiungere oggi una piena maturità che lo colloca fra i linguaggi più usati dell'ultimo anno. Proprio uno dei framework più noti sul web: Ruby On Rails ha vinto l'anno scorso il premio come miglior framework per lo sviluppo internet esistente, a dispetto di molti colossi dell'informatica che operano in questo campo ed a testimonianza della grande maturità che questo linguaggio ha raggiunto. La sintassi è semplice, il linguaggio elegante, completo e particolarmente versatile, la curva di apprendimento molto bassa. Si tratta di uno strumento rapido che può coadiuvare lo sviluppo tradizionale e perché no? in molti casi sostituirlo.

Directory:/Ruby

PYTHON 2.5

L'EX GIOVANE RAMPANTE

Python è stato considerato per lungo tempo il nuovo che avanza. Attualmente non lo si può più definire in questo modo, Python è ormai un linguaggio stabile e completo che trova applicazione inun gran numero di progetti. Se ne parla sempre di più in campo industriale come su Internet. Soprattutto un gran numero di applicazioni anche in ambiente Windows girano ormai grazie a Python e presentano interfacce grafiche ottimamente strutturate. Ciò nonostante Python rimane un grande linguaggio di scripting adatto a gestire in modo completamente automatico buona parte di un sistema operativo sia esso Linux o Windows

Directory:/Python

SNIPPET COMPILER 2.0

PER COMPILARE UN PICCOLO PEZZO DI CODICE

Un utility che consente di scrivere, compilare e fare girare piccoli spezzoni di codice. Molto utile se si vuole provare il funzionamento di una porzione di codice senza per questo volere creare un completo progetto con Visual Studio prima di eseguire il tutto

Directory:/Snippetcompiler

ULTIMATE ++ 2007

L'IDE QUASI RAD PER C++

Se DEV C++ rappresenta ormai uno standard per i programmatori C++, Ultimate ++ si sta dimostrando un'ottima alternativa. Leggero, veloce, dotato di alcune estensioni che lo rendono in parte RAD, viene distribuito con il compilatore MinGW. E' dotato naturalmente di tutte le caratteristiche di un buon IDE, tuttavia dispone di un'organizzazione che lascia rapidamente intendere che da questo ambiente dovremo attenderci delle grosse novità in futuro, sia in termini di prestazioni che di completezza

Directory:/ultimatepp

PIRCBOT 1.4.4

UN BOT IRC PROGRAMMABILE IN JAVA

Interessante questa idea di fornire agli amanti di IRC una serie di API che consentono di programmare un bot utilizzando Java. Normalmente chi si dedica a questo tipo di applicazioni utilizza Eggdrop e programma in TCL, in tutti e due i casi non si tratta di tecniche di larga diffusione, viceversa Java è un linguaggio che gode di un'immensa popolarità, per cui il poter programmare dei bot attraverso la flessibilità di Java rappresenta senza dubbio un'opportunità interessante

Directory:/Pircbot

JAMES 2.3.0

IL MAIL SERVER DI APACHE

| Completamente scritto in Java è dotato di tutte le caratteristiche essenziali di un ottimo mail server- Gestisce SMTP e POP3, filtri blacklist ed è sufficientemente performante da poter essere utilizzato in fase di produzione. Quel che più conta è estendibile facilmente controllabile e personalizzabile da chi ne fa uso

Directory:/James

ICECAST 2.3.1

IL SERVER DI STREAMER OPENSOURCE

Avete clienti che vi hanno chiesto di portare la propria radio su Internet? Più semplicemente volete provare a mettere in piedi una radio che trasmetta solo sul web. Icecast almeno dal punto di vista software, è quello che fa per voi.Avete bisogno di un encoder che prenda il suono dalla scheda audio, lo trasmetta al server Icecast e il gioco è fatto Qualunque client connettendosi al server icecast potrà ascoltare la vostra musica

Directory:/Icecast

SCOPRIAMO GLI AUTOMI CELLULARI

DAI PADRI DELL'INFORMATICA ECCO UNA TEORIA AFFASCINANTE CHE MODELLIZZA, IN QUALCHE MODO, IL COMPORTAMENTO DELLE CELLULE E LE INTERAZIONI CHE HANNO CON LE ALTRE PER RISOLVERE PROBLEMI IN MODO QUASI "UMANO"

Gli automi cellulari, meglio conosciuti nell'accezione inglese cellular automa o semplicemente CA, come li indicheremo in seguito, sono stati introdotti da un personaggio ben noto, addirittura il padre del computer, John Von Neumann, che sviluppò il modello negli anni cinquanta insieme a Stanislaw Ulam. Una definizione che fa capire bene ed in modo diretto di cosa stiamo parlando è proposta in Wolfram Mathworld secondo cui *un CA è un insieme di unità o celle "colorate" su di una griglia di una ben definita forma che evolvono attraverso un numero finito di scansioni temporali in funzione di regole di vicinanza alle celle*. Si tratterà di capire cosa si intende per celle, per griglie e per regole di vicinanza. Ad ogni modo prima di entrare nei particolari si può già notare un'analogia con gli algoritmi evolutivi dove le varie celle possono essere considerati membri di popolazione che si riproducono e si adattano all'ambiente. Certo in questo caso non è presente la mutazione ma si può individuare una sorta di crossover al momento della riproduzione. Nello spazio a disposizione cercheremo di esplorare il maggior numero di CA, ovviamente prediligeremo i più interessanti.

DEFINIZIONE DI CA

In sintesi i CA sono caratterizzati dalle seguenti proprietà:

- La geometria della matrice delle celle;
- L'intorno della cella e dal concetto di adiacenza;
- Il numero di stati per cella;
- Le funzioni o regole di transizione per il passaggio di stato di una cella;

La geometria della matrice delle celle può essere a qualsiasi dimensione. Nella prima applicazione che esamineremo, ad esempio, sarà di dimensione 1. I più comuni CA sono comunque dimensione 2. L'intorno di una cella può comprendere le celle fisicamente adiacenti oppure le celle determinate tramite una funzione metrica (distanza) definita nello spazio delle celle. A tale proposito si possono individuare diversi tipi di intorni. Gli automi cellulari binari sono caratterizzati da due soli stati per cella (1 o 0). Altri CA, usati ad esempio per sistemi dinamici complessi, prevedono anche un numero elevato di stati. Il numero di regole necessarie per stabilire il prossimo stato di una cella cresce esponenzialmente rispetto al numero dei possibili stati della cella.

Per chi volesse affrontare in modo rigoroso l'argomento può fare riferimento alla definizione formale di CA che si trova in letteratura, essa prevede una quadrupla < d, Q, N, f > in cui:

1. d è un numero intero positivo, detto dimensione;
2. Q è un insieme finito, detto spazio degli stati;
3. N è un sottoinsieme finito di Z^d, detto indice di vicinato;
4. f è una funzione definita su $Q^{|N|}$ a valori in Q tale che, detto c_i^t lo stato della cella nel punto i dello spazio Z^d al tempo t, e indicati con $n_1, n_2, ..., n_{|N|}$ gli elementi di N, risulta:
   $c_i^{t+1} = f(c_{i+n_1}^t, c_{i+n_2}^t, ..., c_{i+n_{|N|}}^t)$

in ogni punto i e ad ogni istante di tempo t.

CA UNI-DIMENSIONALI DI WOLFRAM

Visioniamo i particolari CA introdotti da Stephen Wolfram. La geometria su cui evolve la popolazione è una retta discretizzata, ossia le unità giacciono su valori interi di una retta. Gli stati ammissibili per ogni singola unità di spazio sono solo due: zero e uno o vita e morte o ancora acceso e spento; il CA è quindi binario. L'intorno di un'unità sono le due celle ad essa affiancate (a destra e sinistra). Si tratta solo di una possibilità, altri CA hanno tre e quattro cellule adiacenti. Lo stato in posizione x al tempo t, dipenderà dallo stato delle celle al tempo t-1 (memoria 1) in posizione x-1, x e x+1 (loca-

CA E TV

In un episodio della serie TV NUMB3RS, fortunato intreccio di crimine e matematica, vengono menzionati i CA ad una dimensione.

lità). Le celle che influenzano uno stato sono tre valori binari, la cella in oggetto e le due adiacenti. Otto saranno le configurazioni diverse (2 simboli elevato a tre valori). Una regola o legge di produzione sarà una sequenza di otto valori binari che indicano lo stato che dovrà assumere la cella x a seconda di quale delle otto configurazioni assume la terna di celle. La sequenza di otto numeri binari (il primo associato alla configurazione acceso, acceso acceso, il secondo relativo alla sequenza acceso, acceso, spento e così via) formerà a sua volta un numero di otto bit ovvero un byte ovvero un numero intero compreso nell'intervallo [0..255]. In conclusione una regola o legge di produzione non sarà altro che un numero. Nella tabella riportata di seguito vi è un esempio.

Terna di celle adiacenti	Regola
acceso, acceso, acceso	spento (0)
acceso, acceso, spento	acceso (1)
acceso, spento, acceso	acceso (1)
acceso, spento, spento	acceso (1)
spento, acceso, acceso	acceso (1)
spento, acceso, spento	spento (0)
spento, spento, acceso	spento (0)
spento, spento, spento	spento (0)

Il numero che risulta dalla regola applicata, considerando che il bit in alto sulla tabella è il più significativo, è 120 nel sistema decimale. Si applica la conosciuta conversione binario decimale.

0*2^0+0*2^1*2^2+1*2^3+1*2^4+1*2^5+1*2^6+0*2^7=
120

Se indichiamo con O spento e con I acceso possiamo vedere come avviene in realtà la transizione. Alcune leggi come: la 0, la 1, la 168 e la 128 conducono rapidamente allo stato di spento di tutte le cellule, qualsiasi sia lo stato iniziale. La legge 255, ad esempio, porta dopo poche iterazioni allo stato di accesso di tutte le cellule. Altre leggi, generano configurazioni sempre nuove che quindi potrebbero considerarsi caotiche; altre ancora sono regolari e producono evoluzioni prevedibili e a volte ripetitive quindi periodiche. La formalizzazione teorica prevede una retta, quindi una linea di lunghezza infinita che implica infiniti automi giacenti su essa, nella pratica la linea viene implementata da un vettore e per generalizzare il concetto di adiacenza il primo e l'ultimo elemento dell'array sono considerati adiacenti; il segmento è nella pratica un anello. In questa situazione in cui di fatto le configurazioni possibili della popolazione sono finite, se ad esempio il vettore ha lunghezza 200 le popolazioni generabili sono 2^200=1,606938044258990275541962092341 2e+60 (un numero di una certa consistenza!!), prima o poi una data popolazione si ripeterà producendo così una ripetizione che innescherà la periodicità. Quindi non si può parlare di movimento caotico puro. In figura 2 sono riportati alcuni esempi di CA uni-dimensionali che hanno popolazione di 200 e numero di generazioni anch'esso di 200; si tratta quindi di matrici 200x200

Precisato che ogni regola di produzione definisce un diverso CA, altre varianti di CA si riscontrano con diverse concezioni di intorno. I più conosciuti propongono tre e quattro celle adiacenti, anziché due come visto. Ovviamente aumenta la complessità computazionale. Esaminiamo i due casi:

- *Tre adiacenti*. Vi sono 4 celle nell'intorno quindi le configurazioni sono 2^4=16 e le leggi possibili 2^16 = 65536;
- *Quattro adiacenti*. Vi sono 5 celle nell'intorno locale e quindi le regole che possono essere prodotte secondo Wolfran sono 2^32=42.949.967.296, dove 32 sono le possibili configurazioni dell'intorno di ampiezza 5. 2^5=32.

	I	I	I				O		
	I	I	O				I		
	I	O	I				I		
	I	O	O				I		
	O	I	I				I		
	O	I	O				O		
	O	O	I				O		
	O	O	O				O		

Fig 1: Regola di produzione 120 per tutti i possibili (8) casi in può trovarsi la cella (colorata turchese).

CLASSIFICAZIONE DI CA

I CA sono stati suddivisi in quattro distinte classi:

- *Classe 1*. Dopo qualche iterazione viene raggiunto lo stato uniforme di spento per ogni cella. È il caso delle leggi 0, 8, 64 o della 224 (in figura).
- *Classe 2*. Si perviene dopo poche iterazioni a evoluzioni periodiche (con periodo molto minore di 2^n, con n dimensione del vettore di stato) o stabili che cioè mantengono sempre gli stessi stati (non necessariamente tutti accesi o spenti). Si tenga presente che questa classe di automi dipende dalle condizioni iniziali. Un esempio tra i tanti è la regola 2.
- *Classe 3*. Le strutture prodotte sono caotiche e sono prive di periodicità, ben inteso con T molto minore di 2^n, si generano quindi figure geometriche come triangoli per i quali però, non vi è alcuna legge di correlazione (almeno non apparente). Esempio di tale classe è la 18.
- *Classe 4*. Vengono generate strutture complesse, spesso irregolari e sensibili alle condizioni

iniziali. La regola più conosciuta di questa classe è la 110.

CA BIDIMENSIONALI

Qui la scena ha come teatro una vera griglia ossia una matrice di celle. Bisogna definire il concetto di intorno. Vi sono due fondamentali modelli per l'adiacienza. Il primo è quello di Moore, per esso una determinata cella ha come adiacenti altre otto, ovvero, tutte quelle che hanno anche un piccolo contatto, come mostrato in figura 4 b. Il secondo modello è quello di Von Neumann che invece considera adiacenti ad una generica cella solo le quattro celle disposte esattamente a nord, est, sud e ovest (figura 4. a).
Il modello di Moore è il più diffuso ed anche stato usato per "the game of life", mentre quello proposto da Von Neumann che cronologicamente è stato il primo ha meno applicazioni. Le leggi di generazione sono un numero davvero immenso, calcoliamone il numero cosi come abbiamo fatto nel caso più semplice dei CA unidimensionali di Wolfram. I possibili intorni di Moore per sistemi a soli due stati (acceso o spento), quindi già semplificati rispetto a sistemi a più di due stati, sono 2^9 = 512, ovvero tutte le possibili disposizioni di 2 elementi della classe 9, dove il nove è il numero che tiene conto della cella in esame più quelle adiacenti. A questo punto per ogni configurazione è necessario abbinare uno stato, ossia un bit, che descrive il cambiamento. Si ottiene così una sequenza di 512 bit, che in altri termini individua una sola regola. Tutte le possibili leggi sviluppabili sono le disposizio-

Fig. 2: CA appartenenti alle quattro classi diverse. a) classe 1,224 regola; b) classe 2, regola 2; c) classe 3, regola 18; d)classe 4, regola 110

ni che possono assumere i bit (quindi due elementi) nella 512 caselle. Per cui in definitiva si ottiene che le regole possibili sono 2^512 che espresso come potenza di 10 diventa 1,3*10^154. Per intenderci un numero con 154 cifre!! Facciamo una breve analisi sul risultato ottenuto. Ad esempio, il numero di regole associato ad un intorno di Von Neumann è sensibilmente minore, infatti con lo stesso procedimento si ottiene: 2^32 ovvero 4294967296, che nonostante le sue dieci cifre in confronto al caso precedente è una quantità "trascurabile". Inoltre, è da ritenersi una semplificazione il considerare per il calcolo sistemi a soli due stati, infatti se aumenta

Fig. 3: Definizione di intorno per CA bidimensionali. A) Intorno di Moore; B) intorno di Von Neumann

il numero di stati di conseguenza aumenta il numero di regole secondo curve esponenziali, così vi lascio immaginare cosa possano significare maggiorazioni di questo tipo per numeri già enormi. Per i CA unidimensionali la regola di produzione era espressa come un numero, il che era attuabile per via dell'intervallo di "congrua" ampiezza, per i CA bidimensionali ciò non è possibile poiché significherebbe basare un programma per computer sull'input di un numero di 154 cifre. Per cui le regole saranno descritte secondo un'altra filosofia che, per così dire, è di tipo descrittivo, in altre parole il cambiamento di stato si avrà quando si verificano delle condizioni che decideremo a priori e che implementeremo come una serie di controlli sulle celle adiacenti. Alcuni esempi proposti nei prossimi paragrafi, chiariscono questo concetto.
È possibile menzionare solo alcuni degli automi cellulari che nel corso degli anni sono stati presentati, tanta ampia e la casistica di riferimento. Il più conosciuto è senz'altro "the game of life" (il gioco della vita) che approfondiremo dopo. Ricerche sul web e particolari riferimenti bibliografici indicano spunti e argomenti correlati ai CA ed altrettante regole di generazione. Esaminiamone qualcuna tra CA a binari (stato acceso o spento). "Tanto per nulla" si basa sulle seguenti regole: ad ogni generazione tutte le celle accese si spengono; l'accensione di una cella si ha quando essa ha due adiacenti attive. Questa regola è particolarmente interessante perché fornisce un elevato grado di duplicazione anche a partire da griglie con pochissime celle accese. "Merletti" è una regola fondata su due norme: una cella rimane accesa se e solo se ci sono almeno quattro celle attive adiacenti; una cella si accende se ha come adiacenti esattamen-

te due celle attive. Trattiamo adesso qualche esempio in cui gli stati delle singole celle sono più di due. In "Brain", CA inventato da B. Silverman vi sono tre stati indicati con morto, vivo e fantasma, le leggi di generazione sono: dallo stato di morto una cella nasce (stato vivo) se ha due celle attive, altrimenti permane nello stesso stato; una cella dallo stato di morto passa a quello di fantasma nella generazione successiva; una cella fantasma passa alla generazione successiva nello stato di morte. Una tale regola fu sviluppata per aumentare al massimo le movimentazioni all'interno del campo. Interessante è il ruolo delle celle fantasma che impediscono l'espansione da entrambi i lati e risulta piacevole l'effetto grafico di "scia" dovuto alla presenza dello stato fantasma. In "Ecolibra", un altro singolare automa cellulare, vi sono ben cinque stati, oltre alla morte e alla vita sono presenti lo stato refrattario e gli stati aggiuntivi di attivo di brain e attivo di life. Tale regola ha lo scopo di simulare un sistema ecologico in cui competono tre tipi di celle di tipo brain e due di tipo "the game of life".

QUANTE DIVERSE REGOLE DI PRODUZIONE CI SONO?

Dall'analisi fatta è possibile estrarre una formula che calcola il numero di regole di produzione Nrp in funzione degli stati che ogni singola cella può assumere, che indichiamo con Ns e delle celle adiacenti più la cella in oggetto, che indicheremo con Nc.

IL GIOCO DELLA VITA

The game of life o semplicemente Life o ancora all'italiana il gioco della vita è la più fortunata delle varianti degli automi cellulari. Trova la sua notevole popolarità nel fatto che è di semplice formulazione, è un ottimo esercizio di programmazione, ed è molto divertente. L'introduzione di tale CA è opera di un'altra nostra conoscenza, più volte nominata tra queste pagine, il matematico J.H. Conway che lo presentò nel 1970. Come per il CA di Wolfram si tratta di individuare delle leggi di riproduzione in funzione dello stato delle celle adiacenti. In particolare si è voluta negare la riproduzione in situazioni di "sovraffollamento" e di "solitudine". L'intorno considerato è quello di Moore (figura 4). Va ricordata la simultaneità della evoluzione. Ad ogni passo tutte le unità che seguono le identiche regole saranno sottoposte alle leggi di produzione (evoluzione) e assumeranno un nuovo stato o permarranno nel vecchio. Le leggi di evoluzione della popolazione sono:

1. Una cella viva che non ha alcun vicino o che ne ha al più uno nella prossima generazione morirà per solitudine.
2. Una cella viva che abbia quattro o più vicini nella prossima generazione morirà per sovraffollamento.
3. Una cella viva che ha due o tre vicini rimarrà in vita nella prossima generazione.
4. Una cella che non abbia vita nasce nella prossima generazione se ha esattamente tre vicini vivi, ne di più e ne di meno. Tutte le altre celle morte rimangono tali nella prossima generazione.
5. Tutte le nascite e le morti prendono posto esattamente nello stesso istante, cosicché le celle morenti non possono partecipare alla nascita di altre celle (nello stato di morte). Così le celle che stanno nascendo non possono partecipare, nello stato di vive alla morte di altre celle.

Le prime due regole definiscono le modalità con cui avvengono le morti che possono essere dovute a solitudine o sovraffollamento. La terza ci indica quando una vita rimane tale nella generazione successiva, mentre la quarta individua le modalità di nascita. Una battuta: mentre per gli esseri reali (e non virtuali) come gli animali o le persone la riproduzione avviene con la compartecipazione tra due, nel gioco della vita sono necessari tre vite. Si ribadisce che la nascita tecnicamente è il cambiamento dello stato di una cella e che una cella che muore in generazioni successive può riprendere a vivere.

Ecco alcuni esempi di popolazioni. Studiamo la loro evoluzione nelle successive generazioni. La popolazione di figura 5a nella prossima generazione si estinguerà per la prima regola. Le celle muoiono per solitudine.

La popolazione rappresentata in figura 5b rimarrà stabile. Per la regola tre, ognuna delle quattro celle avendo tre vicini rimarrà in vita. Non potranno essere inoltre attuate altre regole per la nascita o la morte, per cui la popolazione rimarrà quella iniziale. L'esempio di figura 5c è un caso di oscillazione. La popolazione per le regole 1, 3 e 4, si trasforma nell'aggregato di figura 5d. In particolare la legge 1 decreta il decesso delle celle più esterne, la legge 3 sancisce la continuazione della vita della cella centrale e la legge quat-

ALCUNE VARIANTI

Esistono un gran quantità di varianti dei CA. La distinzione può avvenire per il numero di stati, per le regole di produzione o per le griglie su cui si evolve la popolazione, la cosiddetta geometria. Su questo ultimo elemento si sono sviluppati "simpatiche" varianti. Tra queste quella che mi interessava menzionare, giacché tra queste pagine abbiamo trattato l'argomento in questione sono le griglie esagonali. Qui ogni cella è un esagono che ha quindi sei celle adiacenti.

tro genera le due nuove vite. Nella generazione ancora successiva la popolazione (riferita questa volta alla figura 3d) si evolve in quella di figura 3c (invocando le stesse leggi del caso precedente); e così il processo di evoluzione continua alternando le due configurazioni.

SISTEMI COMPETITIVI

Un interessante CA analogo al gioco delle vita in cui convivono due differenti vite simula sistemi in competizione tra loro. Anche di questa tipologia ne esistono diversi, ne consideriamo uno a titolo di esempio. Ogni unità nella griglia potrà corrispondere a uno tra tre differenti stati, i due di attività delle due differenti vite e quello di inattività o spento che è comune ai due automi. È quindi necessario definire una legge di evoluzione. Tre regole sono sufficienti; ognuna è associata alla transizione di un singolo stato. Si associa un colore ad ogni stato per rendere più comprensibile la legge, ad esempio: nero per inattivo, blu per attività del primo automa e verde per attività del secondo automa. Una cella che sia spenta (nera), se nel suo intorno ha esattamente tre celle blu (attive del primo life) alla generazione successiva diventa blu; se invece una cella inattiva ha nel suo intorno esattamente tre celle verdi alla prossima generazione diventa verde; in tutti gli altri casi essa permane nello stato di inattività. L'intorno è quello di Moore fatto di otto celle. La seconda regola indica che se una cella è blu ossia se è attiva per il primo automa, rimarrà in tale stato se e soltanto se nel suo intorno vi sono due o tre celle nello stesso stato; se invece, vi sono due o tre celle verdi adiacenti allora si trasformerà in questo nuovo stato (attiva del secondo life); infine, se nessuna di queste condizioni è verificata allora la seconda regola impone che la cella diventi inattiva. La terza regola è speculare alla seconda basta, infatti, invertire i due stati di attività dei rispettivi automi, ovvero invertire il verde con il blu e viceversa.

Fig. 5: Esempi di evoluzione di popolazioni per il gioco della vita

RETI NEURALI E AUTOMI CELLULARI

Le reti neurali cellulari (cellular neural networks CNN) sono un modello di elaborazione proposto da Chua e Yang nel 1988 definito come un insieme di circuiti non lineari in uno spazio n-dimensionale con una struttura di elaborazione parallela ed asincrona. Una CNN è un modello di rete neurale in cui ogni unità (cella o neurone) è connessa solo ad altre unità appartenenti ad una zona della rete ad essa contigua detta vicinato o intorno.
Le reti neurali cellulari sono un modello di calcolo che riassume alcune caratteristiche tipiche delle reti neurali e degli automi cellulari. Infatti, una caratteristica delle CNN è la località delle connessioni tra le unità. In questo tipo di reti neurali l'informazione viene scambiata direttamente solo tra unità vicine. Questa caratteristica li rende in qualche modo simili ai CA e li differenzia dagli altri modelli di reti neurali proposti in letteratura.
Il fatto che la comunicazione sia locale non limita le capacità computazionali delle CNN, infatti è stato dimostrato che il modello CNN è universale essendo equivalente ad una macchina di Turing.
Le principali caratteristiche di una rete neurale cellulare sono:

- **Una CNN è una griglia regolare n-dimensionale di elementi detti celle;**
- **Ogni cella costituisce un elemento di elaborazione con più input e un singolo output;**
- **Una cella è caratterizzata da un vicinato e da uno stato interno che in alcuni casi non è osservabile dall'esterno della cella;**
- **I dati ed i parametri di una CNN hanno generalmente valori continui;**
- **Una CNN può operare sia con valori temporali continui sia con valori temporali discreti**
- **Può essere definita più di una rete di connessione tra le celle con differenti dimensioni di vicinato;**
- **Le CNN elaborano per più di una interazione e quindi appartengono alla classe delle reti di tipo recurrent.**

CONCLUSIONI

Si sbaglia se si pensa che i CA siano soltanto un modo per soddisfare il nostro estro di programmatori. Attraverso questo strumento si sono trattati seri studi che hanno risolto diversi problemi. In generale è stato possibile simulare diversi fenomeni naturali. Particolarmente adatto si è mostrato il modello di simulazione dei CA applicato al comportamento dei gas perfetti. Buoni risultati sono stati ottenuti anche nell'emulazione del movimento di filamenti del DNA. Infine, sono da apprezzare i risultati ottenuti nell'analisi dei comportamenti di popolazioni

Fabio Grimaldi